Universitat Politècnica de Catalunya

Projecte de final de carrera

Anàlisi, modificació i extensió d'una aplicació Android aliena sobre seguretat

Autor: Joan Rodrigo Carreras
Director: Manuel Medina Llinàs
Departament d'Arquitectura de Computadors
Facultat d'Informàtica de Barcelona
Juny 2015

DADES DEL PROJECTE

Títol del Projecte:	Anàlisi, modificació i extensió d'una aplicació Android aliena sobre seguretat
Nom de l'estudiant:	Joan Rodrigo Carreras
Titulació:	Enginyeria Informàtica
Crèdits:	37.5
Director/Ponent:	Manuel Medina Llinàs
Departament:	Departament d'Arquitectura de Computadors
Arxius externs:	https://github.com/Joaneet/AutoLatch

MEMBRES DEL TRIBUNAL (nom i signatura)

President: German Santos Boada

Vocal: Jordi Boronat Medico

Secretari: Manuel Medina Llinàs

QUALIFICACIÓ

Qualificació numèrica:

Qualificació descriptiva:

Data:

Taula de Continguts

1. Introducció

El projecte “Anàlisi, modificació i extensió d’una aplicació Android[1] aliena sobre seguretat”. Té com a objectiu, agafar una aplicació Android ja publicada, de tercers, on es desconeix el codi, i aplicar-li una extensió útil, fen-la més completa.

Com que es tracta d’una aplicació de tercers no preparada per ser modificada i de la qual desconeixem el codi, per tal de realitzar l'extensió, caldrà abans buscar una manera d’analitzar una aplicació ja publicada, i ser capaç de modificar-la per tal de poder estendre les seves funcionalitats.

En primer lloc, es descriu el perquè de la tria de l'aplicació. S’analitza i es raonen les seves debilitats, i en concordança a aquestes s'explica quines són les millores que s’incorporaran.

Es fa una breu descripció de la plataforma Android, perquè el lector tingui uns coneixements mínims per seguir les següents explicacions.

Després es passa a fer una anàlisi de com es pot fer enginyeria inversa de l’aplicació, es posa en pràctica els coneixements explicats i fins i tot es descriuen les noves tècniques de l'autor utilitzades.

Una vegada mostrat com podem modificar l'aplicació per estendre-la segons els nostres requeriments, s’explica el procés de desenvolupament d’aquesta extensió.

1.1. Aplicació escollida, Latch, Que és?

Il·lustració 1: Icona de Latch

L'aplicació que pretén estendre aquest projecte és Latch[2]. Latch, principalment, és un servei basat en interruptors o cadenats, aquests interruptors estan associats a un servei, de manera que aquest servei només es pot utilitza quan l'interruptor està obert. Els interruptors poden ser oberts/tancats des d'un dispositiu mòbil.

Així doncs si per exemple associem un d'aquests interruptors, al nostre compte de WordPress[3], no hi podrem accedir fins que no tinguem l'interruptor obert, encara que sapiguem el nom d'usuari i contrasenya. D'aquesta manera donem un segon factor d'autentificació[4], al nostre compte WordPress, i fins i tot en cas que algú utilitzi les nostres credencials per entrar-hi però amb l'interruptor tancat, rebrem una alerta en el nostre telèfon intel·ligent.

Latch és una aplicació relativament nova (té aproximadament un any de vida), i va desenvolupant diversos plugins per associar-se amb tot tipus de serveis.

No només es pot utilitzar com a segon factor d'autentificació de credencials per portals d'internet, sinó que es pot fer servi per activar o desactivar funcionalitats a qualsevol cosa amb connexió a internet.

Per exemple, en un ordinador personal podríem utilitzar Latch per activar/desactivar el login, els ports usb, la connexió a la xarxa, algunes carpetes o unitats, etc.

Les possibilitats són molt diverses, fins i tot hi ha un plugin per controlar l'obertura d'una porta.

Il·lustració 2: Alguns puglins de Latch[5]

Cal destacar, a més, que la majoria de plugins són OpenSource.

1.1.2 Perquè Latch és interessant?

Actualment un dels problemes més greus en la seguretat informàtica (especialment per internet) és el robatori d'identitat.

Tot tipus d'atacs per obtenir les credencials dels usuaris són duts a terme dia a dia. Tècniques com el fishing[6], sniffing[7][8][9], suplantació dns[10][11], atacs a routers[12], exploits[13], suplantació d'aplicacions o tot tipus de malware[14], són utilitzades entre moltes altres per acabar robant les credencials d'usuari.

Aquestes credencials poden ser utilitzades pel mateix usurpador o venudes a tercers. No és poc freqüent veure notícies on fins i tot milions de credencials d'usuari son publicades a internet.

Per exemple el setembre de 2014 va aparèixer la notícia que en un fòrum rus es podia trobar un arxiu amb més de 5 milions de credencials d'usuari.

https://geekytheory.com/cambia-tu-contrasena-de-gmail-puede-que-te-la-hayan-robado/

Latch intenta afrontar aquest problema, donant un segon factor d'autentificació a més del ja establert. De manera que tot i que les nostres credencials caiguin en males mans, i un malfactor tingui per exemple el nostre nom d'usuari i contrasenya per un servei concret, no podrà accedir gràcies al fet que es trobara bloquejat per Latch. I en aquest cas, a més, Latch ens alertarà de què una altra persona intenta accedir al servei, poden d'aquesta manera agafar les mesures que trobem oportunes.

1.1.3 Debilitats de Latch?

Com ja he dit Latch ens permet principalment activar o desactivar cert servei. Segons el meu punt de vista, el problema principal és que l'usuari és per defecte mandrós, fins i tot a l'hora de cuidar les seves credencials.

Per exemple, es recomana que la contrasenya sigui difícil d'endevinar, però els usuaris acostuma'n a posar contrasenyes fàcils (tot i que la majoria de serveis obliga a certa complexitat, no acceptant per exemple contrasenyes sense números, majúscules o minúscules).

http://antoniogonzalezm.es/50-contrasenas-mas-utilizadas-top-50-passwords-y-brutus-remote-password-craker/

També es recomana posar una contrasenya diferent per a cada servei (ja que si un es compromet, quedarien compromesos tots els nostres serveis) però l'usuari acaba fent cas omís i posa la mateixa contrasenya a tot arreu (contrasenya a més a més relativament senzilla).

Extrapolant aquests exemples a l'ús de Latch, la idea és bona, però el que l'usuari acabaria fent es deixar tots els serveis oberts, potser perquè s'oblida de tancar-los o perquè al final fa mandra anar obrint i tancant, fent que Latch poques setmanes després deixi de ser útil per la majoria dels usuaris.

1.1.4 Com es pot millorar Latch

La solució que jo proposo, és automatitzar el tancament/obertura dels serveis. Latch és una aplicació que es fa servi des d'un dispositiu mòbil, i aquest sap moltes coses sobre nosaltres. La meva idea és utilitzar la informació que ens pot aportar el nostre dispositiu mòbil per controlar els panys que Latch ens ofereix.

Per exemple, aprofitant que el nostre telèfon intel·ligent coneix on ens trobem, podem automatitzar Latch perquè un cert servei només estigui disponible quan nostres ens trobem a un cert lloc. Imaginem que el correu de la feina el consultem normalment només quan ens trobem en el nostre despatx, podríem automatitzar l'obertura/tancament del nostre correu de la feina quan estiguem en el despatx gràcies al fet que el telèfon intel·ligent sap quan i som.

Cal destacar que no és necessari tenir el posicionament GPS activat per tal de què el telefono mòbil sàpiga on ens trobem, sinó que també ho pot saber mirant a quina xarxa wifi o bluetooth ens trobem connectats.

Un altre exemple seria un servei que nomes utilitzem des del dispositiu mòbil, podríem bloquejar el servei quan el mòbil estigui bloquejat i desbloquejar-lo en cas contrari.

Automatitzar el procés ens pot arribar a donar moltes possibilitats i més, gràcies a la versatilitat de poder connectar Latch a tot tipus de servei. Altres exemples més curiosos podrien ser:

- Desactivar ports USB quan se'm desconnecti la connexió bluetooth del portàtil, perquè ningú em pugui robar les dades del portàtil o instal·lar un virus.

- Permetre l'obertura de la porta del pàrquing quan em trobi a 20 metres de casa.

- Tancar la connexió a internet de casa meva quan no hi sigui.

- No permetre la lectura del disc dur del meu ordinador mentre no hi estigui a la vora (no estigui connectat amb bluetooth).

2. Android

La plataforma utilitzada per l'aplicació es Android, com ja hem dit prèviament volem fer una modificació d'una aplicació ja llançada. Per tant ens cal una comprensió total (de baix nivell) de com funcionen les aplicacions en el Sistema Operatiu Android.

2.1 Què és Android?

Android és un sistema operatiu basat en el nucli Linux[15][16]. Inicialment va ser dissenyat principalment per dispositius mòbils i pantalles tàctils, més endavant va abastar altres dispositius, com rellotges intel·ligents, televisors i automòbils entre d'altres.

Inicialment va ser desenvolupat per Android Inc., empresa a la que Google va oferir suport econòmic, però més tard, el 2005, va comprar.

Android va ser presentat el 2007 de la mà de la fundació Open Handset Alliance[17] (un consorci de companyies de hardware, software i telecomunicacions). El primer Telèfon intel·ligent a utilitzar Android va ser el HTC Dream que es va vendre l'octubre de 2008.

Android a diferència d'altres SO per dispositius mòbils, com podrien ser iOS[18] o Windows Phone[19], es desenvolupat de forma oberta, el que significa que es pot accedir al seu codi font (http://source.android.com), i fins i tot a la seva llista d'incidències (https://code.google.com/p/android/issues/list).

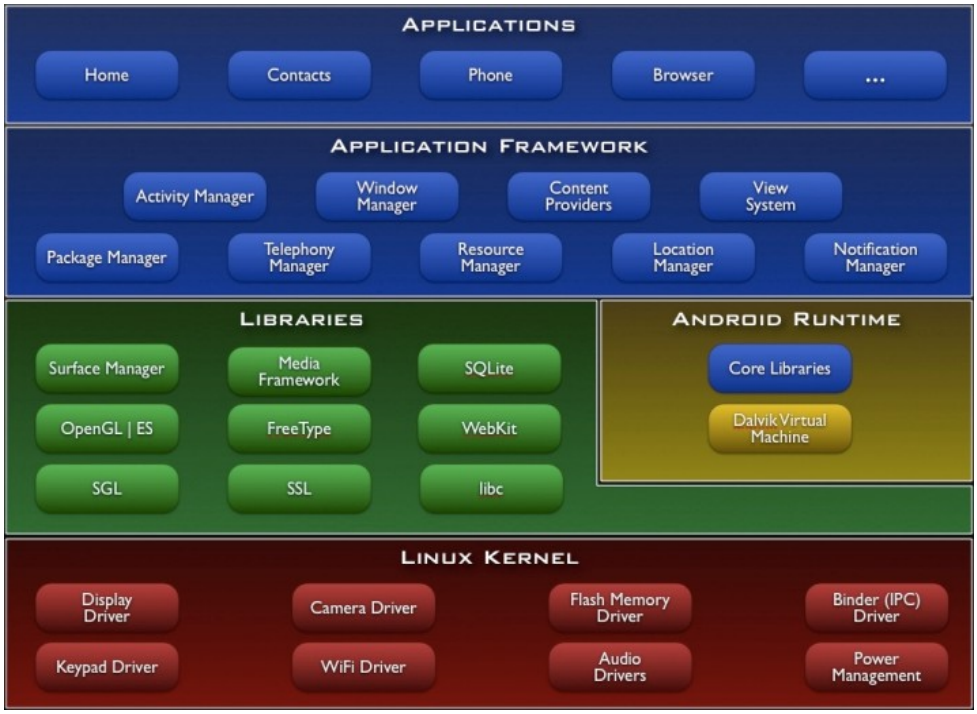
Cal especificar però, que no tots els dispositius mòbils utilitzen el mateix codi Android, ja que la companyia del telèfon intel·ligent, pot fer-hi modificacions, o no escollir la ultima versió disponible. Fins i tot entre models de la mateixa companyia és habitual veure que no tots utilitzen la mateixa versió del SO Android.

Versió	Nom	Data	API	Quota
5.1	*Lollipop*	06 d'Abril 2015	22	0,7%
5.0	*Lollipop*	3 de Novembre 2014	21	9,0%
4.4	*Kit Kat*	31 d'Octubre 2013	19	39,8%
4.3	*Jelly Bean*	24 de Juliol 2013	18	5,5%
4.2	*Jelly Bean*	13 de Novembre 2012	17	18,1%
4.1	*Jelly Bean*	9 de Juny 2012	16	15,6%
4.0	*Ice Cream Sandwich*	16 de Desembre 2011	14	5,3%
2.3	*Gingerbread*	9 de Febrero 2011	10	5,7%
2.2	*Froyo*	20 de Maig 2010	8	0,3%

Taula 1: Versions d'Android (3-10-14)

2.2 Arquitectura d'Android

Android és un sistema operatiu modern, on des d'un primer moment, la seguretat és un factor primordial a l'hora de desenvolupar l'arquitectura[20][21][22]. Des d'aquest punt de vista, Android està dividit en capes, on es té en compte la seva posició. Les capes només poden comunicar-se amb les altres capes adjacents i utilitza els serveis de la seva capa inferior. Les capes pressuposen que les altres capes són segures, de forma que nomes s'ha de preocupar per la seva pròpia seguretat.



Il·lustració 3: Arquitectura de capes Android

Els components blaus, estan escrits en Java[23], i s'executen dintre de la màquina virtual Dalvik[24].

Els components verds i vermells estan escrits en C o C++.

Actualment, la part del kernel té suport per a 32 i 64 bits, mentre que la resta de moment només s'executa en 32 bits.

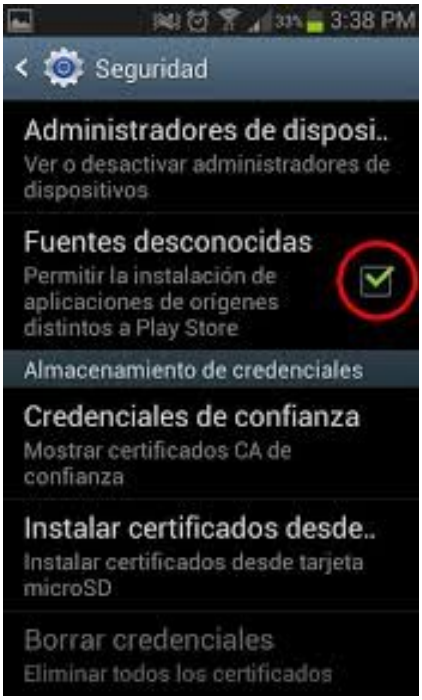
2.2.1 Breu introducció a les Aplicacions d'Android

Per norma general, les aplicacions es desenvolupen amb el llenguatge de programació Java, amb el kit de desenvolupament de software Android SDK[25] [26], però també hi ha disponibles pels desenvolupadors altres eines, on s'inclou, un kit de desenvolupament natiu[27] per aplicacions o extensions en C o C++ i Google App Inventor[28], un entorn visual per programadors poc experimentats.

Les aplicacions normalment vénen empaquetades en un arxiu APK[29], el qual Android manipula i facilita la instal·lació.

La forma habitual d'instal·lar noves aplicacions és a través de l'aplicació Play Store, la qual es troba per defecte en la majoria telèfons intel·ligents amb Android. Aquesta aplicació ens ofereix un mercat de noves aplicacions, Google Play[30], les quals podem descarregar i instal·lar en poques interaccions. Fins i tot ens ofereix la possibilitat de veure comentaris i puntuacions d'altres usuaris sobre la App.

Android a més, de moment, ens permet instal·lar una nova aplicació des d'altres markets, o fins i tot descarregar-la des de qualsevol adreça d'internet.
Tot i que per fer-ho s'ha d'activar l'opció de permetre instal·lar fonts desconegudes[31]:



Il·lustració 4: Menú per activar fonts desconegudes

Una vegada descarregada el paquet d'aplicació (arxiu APK), abans de procedir a la instal·lació, Android, ens mostra quines accions potencialment perilloses podria realitzar, veient quins permisos està requerint. I l'usuari amb aquesta informació pot decidir si continuar amb l'instal·lació o l'atura.

2.2.2 Els arxius APK

Com ja hem comentat abans, una aplicació és empaquetada en un arxiu APK, que Android utilitza per realitzar-ne la instal·lació. Aquest arxiu no és més que un arxiu comprimit ZIP[32] i signat.

Com que es tracta d'un arxiu comprimit ZIP, podem utilitzar el nostre gestor preferit d'arxius ZIP (7-Zip, Winzip o WinRAR) per obrir-lo i veure que hi ha.

Habitualment els arxius que ens podem trobar son:

- Un arxiu classes.dex[33], el qual conte el codi del programa en Dalvik Bytecodes.

- Un arxiu AndroidManifest.xml[34], on hi ha les directrius per què Android pugui executar correctament l'aplicació.

- Un arxiu resources.arsc[35], que conté els pre-compilats.

- Una carpeta res, on hi ha els recursos de l'aplicació.

- Una carpeta assets, on hi ha els arxius de suport de l'aplicació.

- Una carpeta Lib, on hi ha les llibreries que necessita l'aplicació, aquesta carpeta pot tenir subcarpetes dependents del processador pel qual la llibreria ha sigut compilada.

- Una carpeta META-INF, on hi han els arxius corresponents al certificat de la firma de l'aplicació.

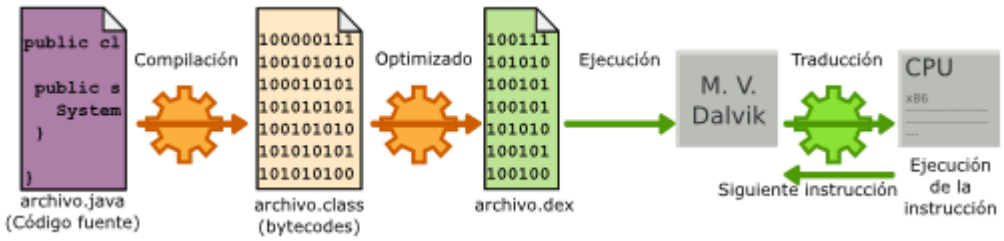
2.2.3 La màquina virtual Dalvik

Com acabem de veure, en l'arxiu classes.dex tenim el codi del programa en Dalvik Bytecodes[36], aquests bytescodes són executats per la màquina virtual Dalvik.

Aquesta màquina virtual, va ser dissenyada per Dan Bornstein, amb la participació d'altres enginyers de Google. Es tracta d'una màquina virtual Java redissenyada que interpreta opcodes diferents, amb l'objectiu, que sigui més eficient en sistemes on la memòria, el processador i l'emmagatzematge siguin realment escassos.

Un altre dels objectius de la màquina virtual Dalvik, és que el SO ha de poder executar diverses instàncies d'aquesta de forma eficient.

Quan compilem el nostre codi java per a ser executat a Android, el que fem és traduir-lo a codi interpretatiu Jasmini[37], aquest es transforma en Bytecodes especials per la màquina Dalvik. I tot el conjunt resultant és empaquetat dintre de l'arxiu .dex.



Il·lustració 5: Transformacions de codi

2.2.4 Els recursos de l'aplicació

La carpeta res d'un arxiu apk, conté els recursos de l'aplicació, aquests normalment són interfícies gràfiques o altres elements i estan escrits en llenguatge XML[38].

XML (eXtensible Markup Language), és un llenguatge de marques desenvolupat per World Wide Web Consortium (W3C) és utilitzat principalment per estructurar informació.

En la il·lustració 6 podem veure com és utilitzat per definir una interfície amb tres botons.

Tot i trobar aquests arxius dintre de la carpeta res del apk, aquests no poden ser directament llegits, ja que estan precompilats per ser interpretats més eficientment per Android. Més endavant veurem com poder visualitzar-los correctament.

Nota : En el punt 4.2.5.2, hi ha una explicació més extensa sobre els recursos de l'aplicació.

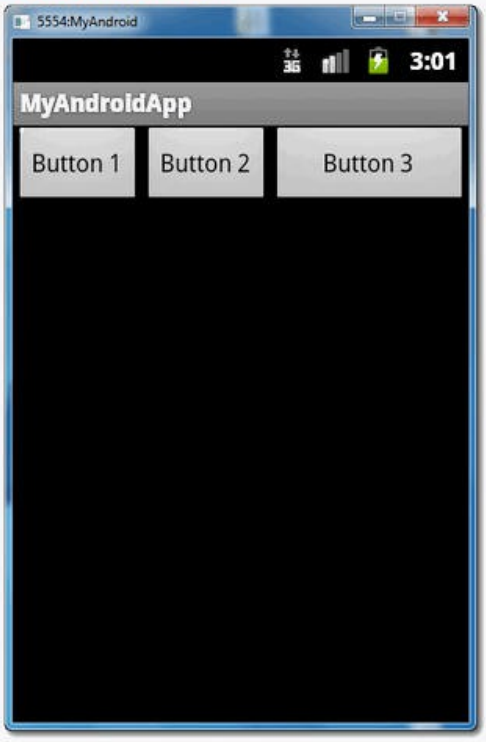
```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal" >

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 1" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 2" />

    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 3"
        android:layout_weight="1"/>

</LinearLayout>
```



Il·lustració 6: Exemple d'interficie definit amb XML

2.2.5 Firmes en les aplicacions

Per instal·lar una aplicació a més, es requereix que aquesta estigui firmada[39]. Un dels avantatges que l'aplicació estigui firmada és que podem tenir informació sobre l'origen de l'aplicació (tot i que res garanteix que la informació sigui certa). Però l'avantatge més important de tenir les aplicacions firmades és que podem comprovar que dues apk diferents provenen del mateix desenvolupador, mitjançant la comparació dels certificats.

D'aquesta manera si volem actualitzar una aplicació, Android comprova que la apk que substituirà la de la versió anterior, tinguin el mateix certificat, en cas contrari Android impedeix l'actualització.

Si el nostre dispositiu està configurat per utilitzar el mode debug, Android ens permet firmar les aplicacions amb un certificat de debug, això és utilitzat quan l'aplicació es troba en procés de desenvolupament. Però quan es vol publicar l'aplicació és necessari crear el nostre propi certificat per a publicar-la.

3. Enginyeria inversa en Android

Per tal de modificar una aplicació de tercers necessitem poder veure el codi de l'aplicació, aquest procés d'obtenció del codi s'anomena enginyeria inversa[40].

Cal especificar a més que el codi obtingut no és el codi amb el qual ha treballat el desenvolupador (en el cas que ens ocupa codi Java). Sinó que per norma general s'obté el codi que és executat d'una manera humanament interpretable.

Així com quan obtenim el codi d'un executable en Windows, és codi assemblador. Quan ho fem per Android el que podem obtenir és codi Jasmin o si el traduïm una mica codi Java (però no és el codi amb el qual ha treballat el desenvolupador).

3.1 Eines disponibles

Actualment tenim diverses eines per realitzar enginyeria inversa en aplicacions Android disponibles. En aquest apartat seran exposades les més eficients i gratuïtes.

3.1.1 dex2jar

http://sourceforge.net/projects/dex2jar/

Es tracta d'un conjunt d'eines open source, escrites en llenguatge java, que tenen com a objectiu ensamblar/desensamblar aplicacions Android i fins i tot firma-les.

Les eines s'utilitzen per consola, i com que estan escrites en java, el desenvolupador ens permet que les puguem utilitzar en diversos sistemes operatius, per això en la versió 0.0.9.15 hi ha scripts diferents segons el tipus de sistema operatiu amb el qual treballem. Així doncs si ens trobem en una

plataforma Windows tenim els arxius bat[41] per poder utilitza les eines, i si ens trobem en Linux tenim arxius sh[42].

Els scripts més interessants a utilitzar són els seguents:

- **dex2jar** Transforma un arxiu dex a un arxiu jar. Els arxius jar són un empaquetat d'arxius class, i els arxius class son bytecode que poden ser executats en una màquina virtual Java. Així que aconseguim tenir bytescode Java els quals són més antics i més estudiats pel món de l'enginyeria inversa, això significa que podem utilitzar eines ja desenvolupades per veure aquest tipus de codi (com JD Project punt 3.1.3).

- **d2j-dex2jar** Versió més moderna de dex2jar, ens permet el mateix però fer-ho directament des d'un arxiu apk.

- **d2j-jar2jasmin** Ens permet obtenir un directori amb arxius de codi Jasmin d'un arxiu jar. El qual hem pogut obtenir amb les eines anteriors.

- **d2j-jasmin2jar** Empaqueta un directori amb arxius de codi Jasmin a un arxiu jar.

- **d2j-asm-verify** Comprova que el codi d'un arxiu jar és correcte i que es pot executar.

- **d2j-jar2dex** Transforma un jar a un arxiu dex.

- **d2j-apk-sign** Firma un arxiu apk.

Com podem veure, utilitzant tot el conjunt d'eines anteriors, i en el mateix ordre, podríem a partir d'un arxiu apk, obtenir el seu codi Jasmin, modificar-lo, i obtenir un nou arxiu apk amb el codi modificat.

Els únics passos que falten, i que es fan amb una eina diferent, són els de modificació del codi Jasmin, és pot fer utilitzant un editor de textos, i el d'obtenir un arxiu apk a partir d'un arxiu dex.

Aquest ultim pas, es fa agafant l'arxiu apk original, i afegim directament el nou arxiu dex obtingut. Ho podem fer-ho amb el nostre gestor preferit d'arxius Zip.

3.1.2 Apktool

http://ibotpeaches.github.io/Apktool/

Eina amb objectius similars a l'anterior, també escrita en Java, ens permet descodificar arxius apk, modificar-los i codificar-los, per tal de modificar una aplicació Android d'alguna manera. Però aquesta eina no esta enfocada a traduir el codi a arxius class[43], per després poder-los veure en visors d'enginyeria inversa convencional Java (que era l'objectiu inicial de dex2jar), sinó que està enfocada a veure i modificar codi Jasmin.

Tot i que dex2jar i apktool poden obtenir codi Jasmin i aquests codis tenen una sintaxi molt similar, no és la mateixa.

La principal avantatge que te respecte a dex2jar és que també ens permet descodificar i codificar els arxius xml que es troben a una aplicació Android.

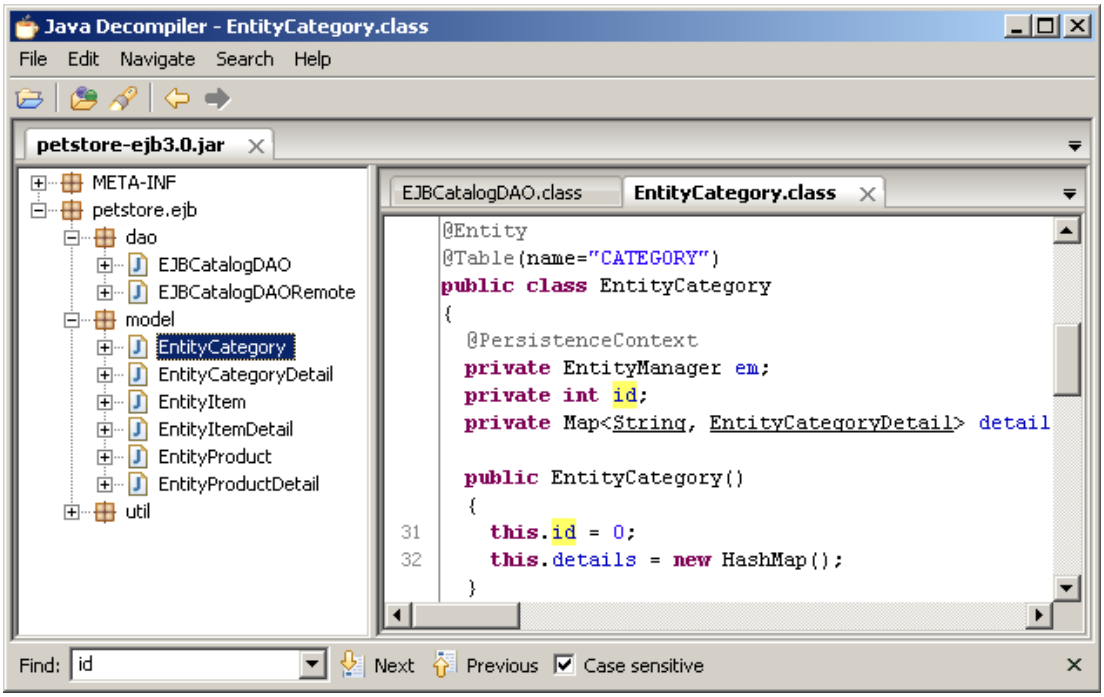
Per descodificar un arxiu apk s'utilitza l'argument d seguit de l'arxiu apk, generant una carpeta amb tots els recursos de l'aplicació Android descodificats.

Per obtenir un arxiu apk a partir de la carpeta generada anteriorment, s'utilitza l'argument b seguit del nom de la carpeta, opcionalment es pot utilitzar l'argument -o seguit del nou nom de l'arxiu apk.

3.1.3 JD Project

http://jd.benow.ca

Per poder veure el codi java d'un arxiu jar, tenim disponible diverses eines, una de les més utilitzades és JD Project, que té una interface GUI[44] per tal d'explorar tot l'arxiu jar[45].



Il·lustració 7: JD Project

3.2 El codi Jasmin

El codi Jasmin que obtenim dels arxius apk, és bastant similar a codi assemblador, per exemple el següent codi en java:

```
public class Calculator extends java.lang.Object {

	public Calculator() {
		super();
		return;
	}

	// centigrade to Farenheit conversion
    // F = 9 * C / 5 + 32
	public float c2f(float local_1) { // note: local_0 = this
		local_1 = 9.0f * local_1;
		local_1 = local_1/5.0f;
		local_1 = local_1 + 32f;
		return local_1;
	}


	// Farenheit to centigrade conversion
```

```
	// C = 5 * (f - 32) / 9
	public float f2c(float local_1) {
		local_1 = local_1 - 32f;
		local_1 = local_1 * 5.0f;
		local_1 = local_1 / 9.0f;
		return local_1;
	}
}
```

En codi Jasmin el veuríem de la següent forma:

```
.class public Calculator
.super java/lang/Object

; default constructor
.method public <init>()V
   aload_0
   invokespecial java/lang/Object/<init>()V
   return
.end method

; centigrade to Farenheit conversion
; F = 9 * C / 5 + 32
.method public c2f(F)F
   .limit stack 4
   .limit locals 2
   fload 1
   ldc 9.0
   fmul
   ldc 5.0
   fdiv
   ldc 32.0
   fadd
   freturn
.end method

; Farenheit to centigrade conversion
; C = 5 * (f - 32) / 9
.method public f2c(F)F
   .limit stack 4
   .limit locals 2
   fload 1
   ldc 32.0
   fsub
   ldc 5.0
   fmul
   ldc 9.0
   fdiv
```

```
    freturn
.end method
```

Com es pot veure a simple vista, és molt més complicat de llegir i entendre el que realment està fent, i codificar codi propi directament pot ser bastant enrevessat.

L'estratègia que jo he utilitzat, per tal de ser més eficient és:

A l'hora d'interpretar codi, no utilitzar el codi Jasmin, sinó transformar l'arxiu dex a jar i utilitzar JD Project.

A l'hora de produir nou codi per l'aplicació, crear un nou projecte Android (utilitzant eclipse[46]), generar un arxiu apk i descodificar-lo utilitzant la mateixa eina que he utilitzat per descodificar l'aplicació Android a la que vull afegir codi, ja que depèn de l'eina, canvia la sintaxi Jasmin, llavors còpia els meus arxius jasmin a la carpeta dels de l'aplicació. I quan produeixi el nou arxiu apk, és produirà amb el meu codi i el de l'aplicació a modificar.

Evidentment a l'aplicació a modificar, serà necessari canviar una mica el codi Jasmin per tal que invoqui el meu codi, la forma més fàcil, és havent produït tot el meu codi en funcions estàtiques[47], per tal que només calgui invocar la meva funció estàtica per executar el codi, n'hi tan sols caldrà instanciar un objecte.

4. Modificant Latch

En aquest apartat vaig a posar fil a l'agulla i explicaré detalladament com modificar Latch per afegir les noves funcionalitats abans descrites.

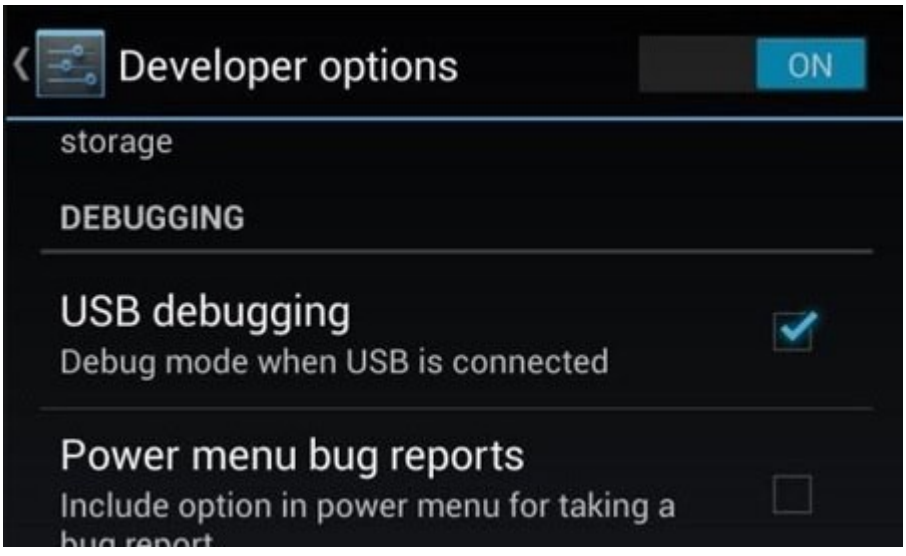
4.1 Obtenint l'Apk

En primer lloc he d'instal·lar l'aplicació (no és un pas necessari però sol ser el més convencional), l'aplicació la trobarem en el Google Play.

https://play.google.com/store/apps/details?id=com.elevenpaths.android.latch

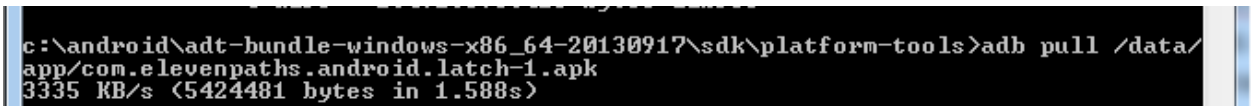
Quan instal·lem una aplicació, ens descarreguem l'apk. Els Apk's que utilitzem es troben al directori /data/app [48][49] i en aquest cas concret es diu com.elevenpaths.android.latch-1.apk, per extreure l'arxiu del nostre dispositiu Android al nostre ordinador, el que farem servi és l'eina adb[50], que es troba en el sdk que Google ens ofereix per desenvolupar per Android.

Una vegada instal·lat el sdk, hem de configurar el nostre dispositiu Android perquè ens permeti utilitzar eines de desenvolupador[51][52], això es fa anant a través del menú configuració, opcions de desenvolupador i activar depuració d'usb (pot variar segons la versió d'Android). Evidentment, caldrà també connectar el dispositiu a l'ordinador mitjançant cable usb.



Il·lustració 8: Menú de desenvolupador

Ara des de la consola de l'ordinador, anem a la carpeta on hàgim instal·lat l'sdk de desenvolupament d'Android hi utilitzem la comanda, adb pull /data/app/com.elevenpaths.android.latch-1.apk



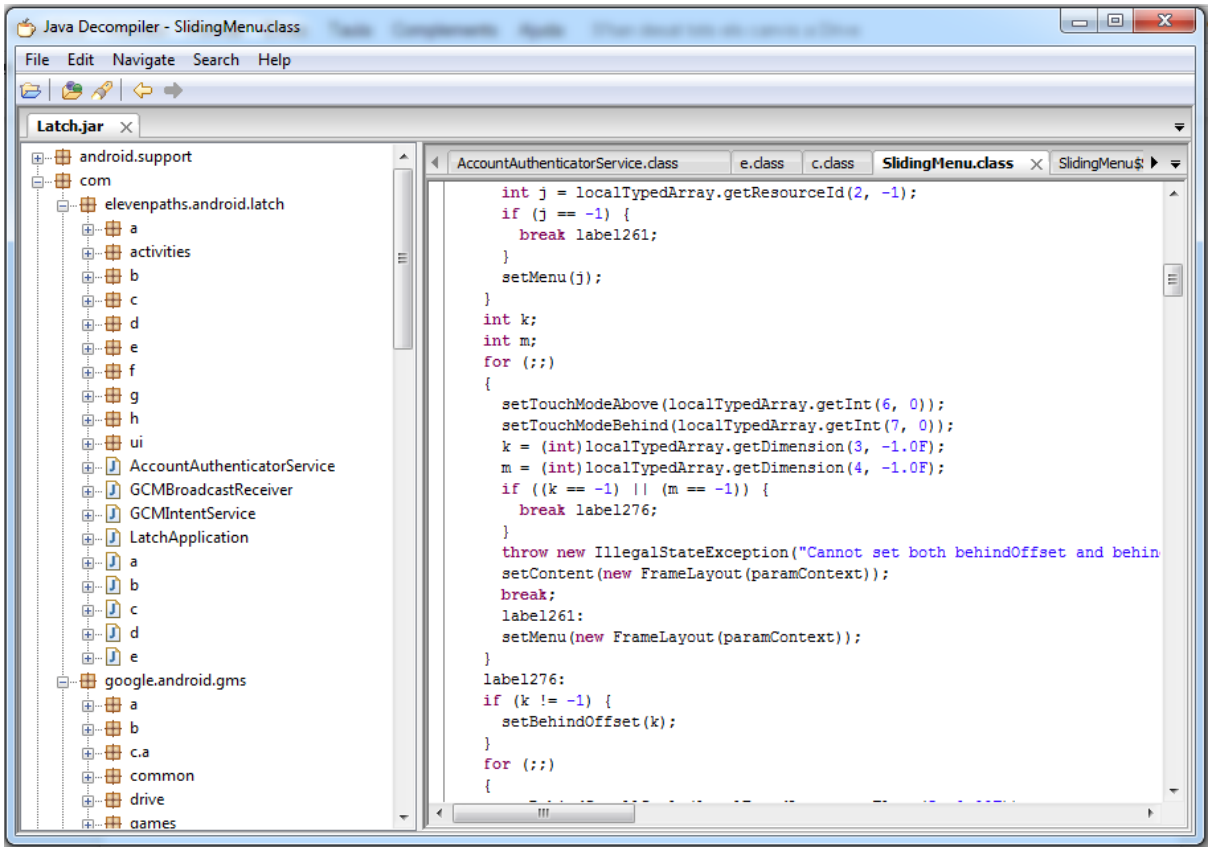
4.2 Anàlisis del codi

Ara que ja tenim l'apk, iniciarem l'anàlisi del codi, per saber de quina forma hem de modificar l'aplicació perquè incorpori les meves millores.

4.2.1 Preparant l'entorn d'analisis

Per fer un anàlisi del codi, com he explicat en el capítol d'eines disponibles, el millor és obtenir el jar a partir de l'arxiu apk, i utilitzar una eina com JD Project per analitzar el codi contingut en el jar.

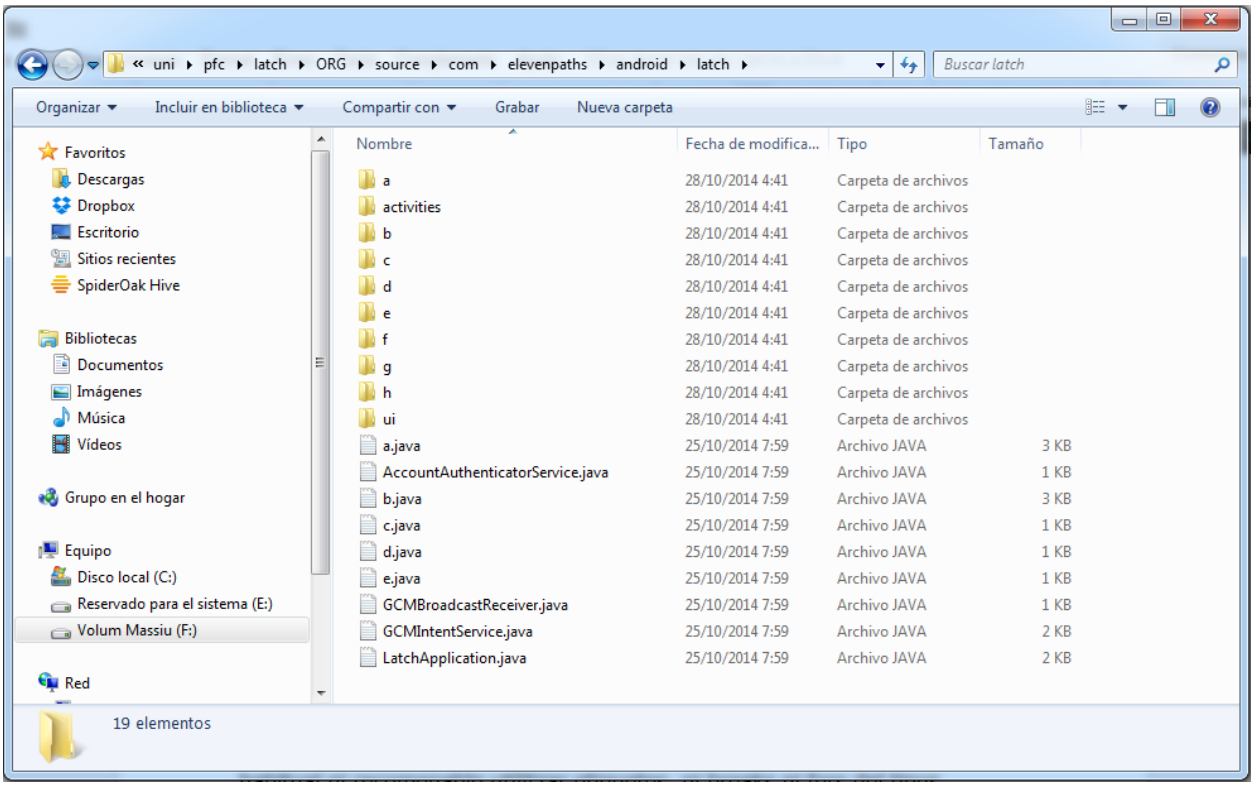
Utilitzem l'eina d2j-dex2jar, contra l'apk, obtenim el jar i després obrint aquest amb JD Project, ja podem veure tot el codi de l'aplicació.



Il·lustració 9: Codi Latch obtingut amb dex2jar

Ja a simple vista, podem veure que el codi es troba ofuscat[53][54], ja que els noms utilitzats en variables, classes, funcions, etc. no són intuïtius a l'hora de desenvolupar, excepte en les classes Estàtiques, ja que si l'ofuscador canvia els noms hi hauria problemes de referències. Les estructures de programació també estan ofuscades, es veu a simple vista, ja que no és ni habitual ni recomanable utilitzar etiquetes, breaks o fors del tipus for(;;;).

En lloc d'analitzar el codi des de Java Decompiler de JD Project, jo prefereixo utilitzar l'opció File -> Save All Resources, que ens permet guarda tots els arxius class de dintre del jar en arxius java dins d'un zip. Després descomprimeixo el zip en una carpeta, i faig l'anàlisi des d'allà. D'aquesta manera puc utilitzar el meu editor de textos preferit o qualsevol altra eina d'anàlisi, que per exemple faciliti la cerca entre arxius.



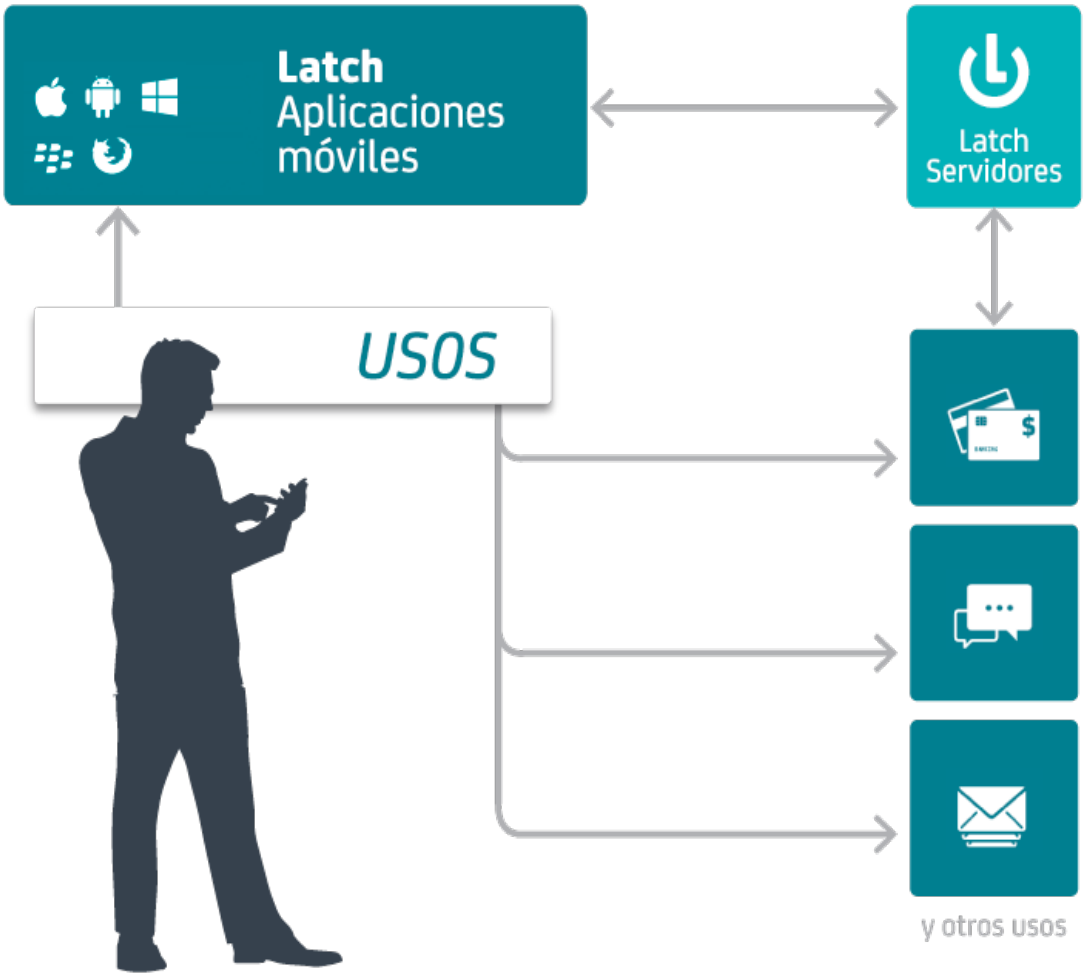
Il·lustració 10

4.2.2 Localitzar l'obertura/tancaments de serveis

Amb el primer que vaig centrar-me és a trobar com Latch obria i tancava els serveis que manipulava, ja que la meva intenció era automatitzar aquest procés, i per tant és la part fonamental de la millora, si no era capaç de trobar com Latch ho feia, era impossible fer les millores que jo pretenia.

Per fer-ho, primer vaig haver d'imaginar-me com està desenvolupada Latch, i després comprovar si aquesta hipòtesi es complia mirant el seu codi.

Quan nosaltres obrim o tanquem un servei, el que fem és enviar una ordre al servidor de Latch, i aquest guardarà la informació sobre el nou estat del servei que hem modificat. Més endavant quan intentem utilitzar aquest servei, el servei es comunicarà amb el servidor de Latch i li preguntarà si el servei està obert o no.



Il·lustració 11: Esquema de funcionament Latch

Així doncs, quan nosaltres obrim o tanquem un servei, no fem res més que connectar-nos al servidor de Latch i enviar-li l'ordre. El més probable és que l'ordre sigui enviada per https[55], perquè és el més fàcil d'utilitzar pel desenvolupador[56], fer una api sobre https esta àmpliament documentat. També per les circumstàncies d'un telèfon intel·ligent és imprescindible utilitzar el port 80 o 443, ja que el desenvolupador no sap la política de xarxa on s'utilitzarà la seva aplicació i aquest port és dels únics que sempre està disponible per connexions de sortida.

Per trobar aquestes peticions intentaré busca si en el codi tenim la paraula clau http, potser els strings a les direccions http/https es troben ofuscats, però el que és més difícil d'ocultar són les classes que continguin el nom http (ja que els

desenvolupadors solen utilitzar llibreries de tercers), que seran les que facin les peticions.

Després de fer la cerca, el que trobo primer és a quines URL es fan les peticions, puc veure els strings a la clase com.elevenpaths.android.latch.d.a (aquests strings per alguna raó no estaven ofuscats).

```
switch (paramInt)
{
default:
  return null;
case 901:
  return new d("https://latch.elevenpaths.com/control/0.9/authenticate", 201, paramList);
case 902:
  return new d("https://latch.elevenpaths.com/control/0.9/pairingToken", 202, paramList);
case 903:
  return new d("https://latch.elevenpaths.com/control/0.9/applications", 202, paramList);
case 904:
  return new d("https://latch.elevenpaths.com/control/0.9/update", 201, paramList);
case 905:
  return new d("https://latch.elevenpaths.com/control/0.9/twoFactorToken", 202, paramList);
case 906:
  return new d("https://latch.elevenpaths.com/control/0.9/logout", 202, paramList);
case 907:
  return new d("https://latch.elevenpaths.com/control/0.9/appPreferences", 201, paramList);
}
return new d("https://latch.elevenpaths.com/control/0.9/appPreferences", 202, paramList);
```

I en segon lloc, també trobo on es fan les crides a aquestes urls, ja que la cerca em permet trobar que es fa ús de la llibreria org.apache.http, i que la clase responsable de fer d'interfície d'aquesta llibreria és com.elevenpaths.android.latch.d.d.

Observant aquesta classe veiem que la funció private HttpResponse a(String paramString), és l'encarregada de fer peticions post a direccions html :

```
private HttpResponse a(String paramString)
{
  Context localContext = LatchApplication.d().b();
  if (this.b) {
    try
    {
      BasicHttpParams localBasicHttpParams = new BasicHttpParams();
      HttpConnectionParams.setConnectionTimeout(localBasicHttpParams, 15000);
      HttpConnectionParams.setSoTimeout(localBasicHttpParams, 30000);
      URL localURL = new URL(this.c);
      URI localURI = new URI(localURL.getProtocol(), "", localURL.getHost(), localURL.getPort(), localURL.getPath(), localURL.getQuery(), null);
      String str = localContext.getPackageManager().getPackageInfo(localContext.getPackageName(), 0).versionName;
      HttpPost localHttpPost = new HttpPost(localURI);
      localHttpPost.setHeader("Cookie", "PATH2_SESSION=" + paramString);
      localHttpPost.addHeader("X-Client-Version", "Android/" + str);
      localHttpPost.addHeader("X-gcm-token", com.elevenpaths.android.latch.h.c.a());
      DefaultHttpClient localDefaultHttpClient = new DefaultHttpClient(localBasicHttpParams);
      if (this.a != null) {
        localHttpPost.setEntity(new UrlEncodedFormEntity(this.a, "UTF-8"));
      }
      HttpResponse localHttpResponse = localDefaultHttpClient.execute(localHttpPost);
      return localHttpResponse;
    }
    catch (Exception localException)
    {
      return null;
    }
  }
  return null;
}
```

I que la funció private HttpResponse b(String paramString), fa crides Get, utilitzant un codi similar a l'anterior funció.

Analitzant més profundament aquesta funció, ens trobem que les dues funciones anteriors de peticions html, són cridades des de la funció public c b() :

```
if (i == 201) {
  localHttpResponse = a(a.a());
}
if (this.d == 202) {
  localHttpResponse = b(a.a());
}
```

I la resposta d'aquestes, és agafada per aquesta funció, i és interpretada en format JSON[57] que és el valor de retorn de la funció:

```
    catch (Exception localException2)
    {
      return new c(800);
    }
    String str2 = localStringBuilder.toString();
    try
    {
      c localc2 = new c(new JSONObject(str2));
      return localc2;
    }
    catch (JSONException localJSONException)
    {
      return new c(800);
    }
  }
  return new c(900);
```

Il·lustració 12: Retorn de la funció

4.2.2.1 Anàlisi dinàmic VS Anàlisi Estàtic

Actualment hem obtingut on es produeixen les invocacions html dins de l'aplicació mitjançant l'ús de l'anàlisi estàtic[58], no hem executat codi, només l'hem analitzat.

En aquest punt interessaria, saber com l'aplicació utilitza aquesta funció, i des d'on és cridada aquesta funció.

Una manera de fer-ho seria continuar l'anàlisi estàtic, buscant aquelles classes que criden aquesta. Però continuar amb aquest mètode pot arribar a ser molt farragós, moltes de les referències que trobem poden no ser les que estem buscant i a més es pot tornar un procés recursiu (quan trobem la classe que l'invoca potser volem saber qui invoca aquesta classe, i per tant tornem a començar).

Una alternativa és utilitzar l'anàlisi dinàmic[59][60], bàsicament consisteix a executar el codi i veure que passa. Però no podem fer ús d'un debug, l'eina en què es basa l'anàlisi dinàmic. Així doncs, he hagut d'idear una altra estratègia.

L'estratègia es basa en l'idea de modificar l'aplicació, executar-la i obtenir un arxiu que ens informi del que ha passat durant l'execució. Per aconseguir aquest objectiu he fet una aplicació Android que conté la següent classe Writer.java:

```
public class Writer{
    private static FileWriter fw;

    public static String Concat(String str1, String str2){ return str1+str2; }

    public static void Write(String str){
        try{
            if (fw==null) fw = new FileWriter("/sdcard/latch.log", true);

            fw.write(str);
            fw.flush();
        }catch (IOException e){ e.printStackTrace(); }
    }

    public static void Write(URL url){ Write("get " + url.toExternalForm() + "\n\t" + url.getQuery() + "\n"); }

    public static void Write(HttpPost post){
        Write(post.getMethod() + " " + post.getURI() + "\n");

        Header[] h = post.getAllHeaders();
        for(int i=0;i<h.length;i++) Write("\t" + h[i].getName() + ": " + h[i].getValue() + "\n");

        try {
            Write("\t Content : " + EntityUtils.toString(post.getEntity()) + "\n");
        }catch (Exception e){}
    }

    public static void WriteThread(){
        AccessController.doPrivileged(new PrivilegedAction<Object>(){
            public Object run(){
                Writer.Write("Thread\n");
                StackTraceElement[] ste = Thread.currentThread().getStackTrace();
                for(int i=0;i<ste.length;i++)
                    Writer.Write("\t" + ste[i].getClassName() + "\t" + ste[i].getMethodName() +
                            "\t" + ste[i].getLineNumber() + "\n");
                return null;
            }
        });
    }
}
```

Com es pot observar, aquesta classe té diferents mètodes estàtics. Els quals escriuen a l'arxiu /sdcard/latch.log.

public static void Write(String str) : Escriu l'string passat.
public static void Write(URL url) : Escriu la url passada.
public static void Write(HttpPost post) : Escriu els arguments posts.
public static void WriteThread() : Escriu les actuals invocacions de pila.

Utilitzant la tècnica breument descrita en el punt 3.2, generem l'apk d'aquesta aplicació Android, la descodifiquem, posem el codi obtingut on tenim el codi de l'aplicació Latch descodificada.

I per exemple si volem obtenir qui crida la funció public c b(), obrim l'arxiu [...]com\elevenpaths\android\latch\d\.d.j el qual conté el codi Jasmin de la classe vista anteriorment i localitzem la funció que ens interessa, veiem que l'inici té el següent aspecte:

```
527  .method public b()Lcom/elevenpaths/android/latch/d/c;
528  .catch java/lang/Exception from L0 to L1 using L2
529  .catch java/lang/Exception from L3 to L4 using L2
530  .catch java/lang/Exception from L4 to L5 using L6
531  .catch java/lang/Exception from L5 to L7 using L6
532  .catch java/lang/Exception from L8 to L9 using L6
533  .catch java/lang/Exception from L10 to L11 using L6
534  .catch org/json/JSONException from L11 to L12 using L13
535  aload 0
536  getfield com/elevenpaths/android/latch/d/d/b Z
537  ifne L14
538  new com/elevenpaths/android/latch/d/c
539  dup
540  sipush 899
541  invokespecial com/elevenpaths/android/latch/d/c/<init>(I)V
542  areturn
543  L14:
544  aload 0
545  getfield com/elevenpaths/android/latch/d/d/d I
546  istore 1
```

Afegim la següent línia a l'inici de la funció:

invokestatic com/megacorp/autolatchplugin/Writer/WriteThread()V

La qual farà que cada vegada que es crida aquesta funció, escrigui en el log, la traça actual de la pila[61], amb el que veurem qui l'invoca.

A continuació, utilitzant dex2jar empaquetem tot el codi i creem el nou arxiu latch.apk, el firmem i instal·lem en el dispositiu Android.

Una vegada instal·lat l'executem i obrim o tanquem un servei dintre de l'aplicació, a continuació mirem que tenim escrit en l'arxiu /sdcard/latch.log, i trobarem una cosa similar a aquesta:

```
c:\android\adt-bundle-windows-x86_64-20130917\sdk\platform-tools>adb shell
shell@android:/ $ cd sdcard
cd sdcard
shell@android:/sdcard $ cat latch.log
cat latch.log
Thread
        dalvik.system.VMStack   getThreadStackTrace     -2
        java.lang.Thread        getStackTrace   591
        com.megacorp.autolatchplugin.Writer$1   run     -1
        java.security.AccessController  doPrivileged    45
        com.megacorp.autolatchplugin.Writer     WriteThread     -1
        com.elevenpaths.android.latch.d.d       b       -1
        com.elevenpaths.android.latch.b.c       a       -1
        com.elevenpaths.android.latch.b.c       doInBackground  -1
        android.os.AsyncTask$2  call    287
        java.util.concurrent.FutureTask$Sync    innerRun        305
        java.util.concurrent.FutureTask run     137
        android.os.AsyncTask$SerialExecutor$1   run     230
        java.util.concurrent.ThreadPoolExecutor runWorker       1076
        java.util.concurrent.ThreadPoolExecutor$Worker  run     569
        java.lang.Thread        run     856
Thread
        dalvik.system.VMStack   getThreadStackTrace     -2
        java.lang.Thread        getStackTrace   591
        com.megacorp.autolatchplugin.Writer$1   run     -1
        java.security.AccessController  doPrivileged    45
        com.megacorp.autolatchplugin.Writer     WriteThread     -1
        com.elevenpaths.android.latch.d.d       b       -1
        com.elevenpaths.android.latch.b.b       a       -1
        com.elevenpaths.android.latch.b.b       doInBackground  -1
        android.os.AsyncTask$2  call    287
        java.util.concurrent.FutureTask$Sync    innerRun        305
        java.util.concurrent.FutureTask run     137
        android.os.AsyncTask$SerialExecutor$1   run     230
        java.util.concurrent.ThreadPoolExecutor runWorker       1076
        java.util.concurrent.ThreadPoolExecutor$Worker  run     569
        java.lang.Thread        run     856
Thread
        dalvik.system.VMStack   getThreadStackTrace     -2
        java.lang.Thread        getStackTrace   591
        com.megacorp.autolatchplugin.Writer$1   run     -1
        java.security.AccessController  doPrivileged    45
        com.megacorp.autolatchplugin.Writer     WriteThread     -1
        com.elevenpaths.android.latch.d.d       b       -1
        com.elevenpaths.android.latch.b.g       a       -1
        com.elevenpaths.android.latch.b.g       doInBackground  -1
        android.os.AsyncTask$2  call    287
        java.util.concurrent.FutureTask$Sync    innerRun        305
        java.util.concurrent.FutureTask run     137
        android.os.AsyncTask$SerialExecutor$1   run     230
        java.util.concurrent.ThreadPoolExecutor runWorker       1076
        java.util.concurrent.ThreadPoolExecutor$Worker  run     569
        java.lang.Thread        run     856
shell@android:/sdcard $ _
```

Il·lustració 13: Obtenció del log

4.2.3 Continuant la cerca

En el resultat del punt anterior, veiem que es crida 3 vegades la funció, i jo només he obert una sola vegada un servei, després de diverses proves (bàsicament miro en quin moment es fan les crides) acabo deduint que només la ultima invocació és la responsable de l'obertura del servei. I mirant la traça veig que la classe que inicia el procés és com.elevenpaths.android.latch.b.g, a partir d'un nou thread, això es veu per què provinc de la funció doInBackgound, la qual és utilitzada per iniciar un nou Thread, cosa perfectament lògica, ja que farem una petició html i no volem que l'aplicació es quedi congelada mentre esperem la resposta de la petició.

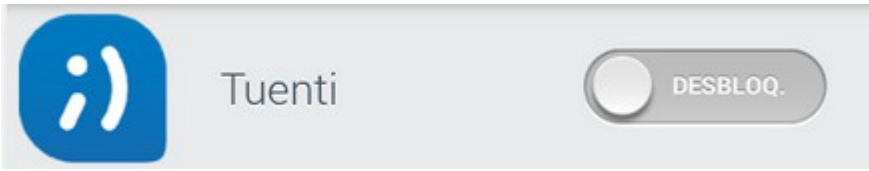
Ara que ja he descrit la tècnica, només es tracta d'anar estirant el fil, i anar veient que és el que trobem.

Així doncs monitorem les crides a com.elevenpaths.android.latch.b.g, anem traçant i finalment obtenim que l'activity[62] que inicia la crida és com.elevenpaths.android.latch.activities.Z_LockAccess amb el següent codi:

```
protected void onCreate(Bundle paramBundle)
{
  super.onCreate(paramBundle);
  setContentView(2130903110);
  ((RelativeLayout)findViewById(2131099768)).setBackgroundResource(2130837596);
  TextView localTextView = (TextView)findViewById(2131099769);
  localTextView.setText(2131361978);
  d.a(localTextView, "fonts/fs_joey_medium.ttf");
  ((Button)findViewById(2131099770)).setOnClickListener(new cm(this));
  ((TextView)findViewById(2131099962)).setText(2131361981);
  ((TextView)findViewById(2131099964)).setText(2131361980);
  String str = getIntent().getExtras().getString("operation_id");
  this.a = LatchApplication.d().a().a(str);
  a();
  if (this.a != null)
  {
    b();
    return;
  }
  LatchApplication.d().a().addObserver(this);
  LatchApplication.d().a().a();
}
```

Com que hem trobat l'activity (sabem que és una activity per què aquesta classe hereda de la classe activity) que llença la invocació podem està segurs que

estem al més alt nivell d'aquesta crida, ja que les activities s'utilitzen per norma general com a interfície amb l'usuari i per tant són les que transformen les ordres de l'usuari en operacions. Concretament si analitzem aquesta activity, ens adonarem que segurament es tracta d'una de les pestanyes per bloquejar desbloquejar serveis, com per exemple aquesta:



Il·lustració 14

Si analitzem la crida subratllada, veurem que es fa a una classe estàtica, i aquesta s'encarrega de construir tot el necessari per portar a terme l'operació, cosa que ens va perfecte. Desglossem la crida per entendre-ho millor:

LatchApplication.d() ens retorna una classe del tipus LatchApplication.

LatchApplication.d().a() ens retorna una classe del tipus com.elevenpaths.android.latch.f.a, aquesta classe el que realment fa és d'interfície amb l'api html, els seus mètodes es transformen en crides a la API html, segons el mètode s'utilitzen diversos paràmetres per fer operacions diferents. La classe acaba cridant a com.elevenpaths.android.latch.b.g.

Si continuem utilitzant la meva classe per monitoritzà, podem veure quins paràmetres fa servi l'aplicació. I per exemple LatchApplication.d().a().a(String str, d.a); str és la id de l'operació (del servei) i d.a és una classe com.elevenpaths.android.latch.c.d la qual s'obté de crides estàtiques al mètode 'a' que ens retorna "on" i el mètode 'b' ens retorna "off".

Per tant ja hem localitzat la part del codi on es produeix el bloqueig, desbloqueig d'un servei.

Per invocar aquesta crida des del meu codi, necessito fer ús de reflection[63], i ho faig de la següent manera:

```
public static void ModifyOperation(String operation, boolean onoff){
    try{
        Object objclass1 = Class.forName("com.elevenpaths.android.latch.LatchApplication")
                .getMethod("d", new Class[0]).invoke(null, null);
        Object objclass2 = objclass1.getClass().getMethod("a", new Class[0]).invoke(objclass1, null);
        Class class1 = Class.forName("com.elevenpaths.android.latch.c.d");

        Field field;
        if(onoff)   field = class1.getField("a");
        else        field = class1.getField("b");

        objclass2.getClass()
            .getMethod("a", String.class, field.get(null).getClass()).invoke(objclass2, operation, field.get(null));
    }catch(Exception e){}
}
```

Vaig obtenint les classes per reflection i les vaig invocant. Depenen de si vull activar o desactivar obtinc una classe o altra de com.elevenpaths.android.latch.c.d tal com ja he descrit anteriorment.

Nota: Com he dit abans ha semblat que hem tingut molta sort, ja que hem trobat crides estàtiques que creen el conjunt de classes per realitzar les operacions. Però realment no ha sigut un cop de sort, ja que és una pràctica habitual programar d'aquesta manera quan necessitem classes auxiliars des de diversos llocs de l'aplicació. A Latch podem bloquejar i desbloquejar un servei des de diversos punts, podem fer-ho per una acció de l'usuari dins de l'aplicació o des d'una alerta rebuda pel núvol, que informa d'un intent d'accés a un servei bloquejat.

4.2.4 Obtenint els serveis de l'usuari

Acabem de veure que ja sabem quin codi cal invocar per bloquejar/desbloquejar un servei, però ens cal saber la ID de l'operació per tal de fer-ho.

A l'Il·lustració 12 ja hem vist que l'API ens retorna l'informació en format JSON, utilitzant Writer.Write(String str), podem logejar l'string str2. En fer-ho he vist que com és previsible, només iniciar l'aplicació el primer que fa és preguntar a l'API quins serveis tenim.

Llavors el que he fet és modificar el final de la crida amb el següent codi Jasmin:

```
;StaticData.receiveData(str2);
aload 11
invokestatic com/megacorp/autolatchplugin/StaticData/receiveData(Ljava/lang/String;)V
```

Com ja es pot veure en el comentari, crido a la meva funció StaticData.receiveData(String data) que conté el següent codi

```
public static void receiveData(String data){ setOperations(data); }

private static void setOperations(String operationsJSON){
    try{
        JSONObject jsonOps = new JSONObject(operationsJSON);
        if((jsonOps.has("data")) && (!jsonOps.isNull("data")) && (jsonOps.getJSONObject("data").has("operations"))){
            operations = new ArrayList<Operation>();
            setOperations(jsonOps.getJSONObject("data").getJSONObject("operations"), null);
        }
    }catch (JSONException e){ e.printStackTrace(); }
}

public static void setOperations(JSONObject opJSON, Operation parent){
    try{
        for(Iterator<String> keys = opJSON.keys(); keys.hasNext();){
            String str = keys.next();
            JSONObject opDesc = opJSON.getJSONObject(str);
            Operation op = new Operation(str, opDesc, parent);
            operations.add(op);
            if (!opDesc.isNull("operations")) {
                setOperations(opDesc.getJSONObject("operations"), op);
            }
        }
    }catch (JSONException e){ e.printStackTrace(); }
}
```

Veiem que manipulo el JSON passat, i omplo l'ArrayList operations amb classes del tipus operation. El JSON que passa l'api ens dóna com a dades importants i que guarda, la id de l'operació, la descripció (el nom que l'usuari veu) i el pare de l'operació (per exemple en un banc, el servei és el banc però les operacións filles a aquest poden ser la de login o transferència).

4.3 Modificació de l'interfície

Per tal que l'usuari de l'aplicació pugui interactuar amb la nostra extensió, he de modificar l'actual interfície de forma que cridi les meves activities.

Així doncs en primer lloc hem de fer enginyeria inversa de la interfície de Latch. Utilitzo l'eina apktool per descompilar l'apk, que produirà entra d'altres la carpeta res, en la qual hi han els resources de l'aplicació escrits en format xml, explicat per sobre a 2.2.4.

4.3.1 Els arxius resource

Principalment els arxius xml de la carpeta res, descriuen 2 tipus de conceptes[64].

En les carpetes layout*, es descriuen com són les interfícies que més endavant utilitzaran les activities i altres classes similars, el que ja havíem vist a 2.2.4.

Després de layout el nom de la carpeta es refereix a quin tipus de pantalla utilitza el dispositiu. Així doncs s'utilitzarà layout-hdpi en pantalles amb un nombre de píxels per polsada considerat alt i layout-mdpi en pantalles amb un nombre de píxels per polsada considerat mitjà.

En les carpetes values*, ens trobem els arxius que contenen els valors per l'aplicació, després de values, el nom de la carpeta, es refereix a l'idioma que fa servir el dispositiu, així doncs els strings a values-es són els utilitzats quan el dispositiu està en castellà i els de values-en quan està en anglès.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="common_google_play_services_install_title">Descargar servicios de Google Play</string>
    <string name="common_google_play_services_install_text_phone">Esta aplicación no se ejecutará si tu teléfono no tiene
    <string name="common_google_play_services_install_text_tablet">Esta aplicación no se ejecutará si tu tablet no tiene
    <string name="common_google_play_services_install_button">Descargar servicios de Google Play</string>
    <string name="common_google_play_services_enable_title">Habilitar servicios de Google Play</string>
    <string name="common_google_play_services_enable_text">Esta aplicación no funcionará si no habilitas los servicios de
    <string name="common_google_play_services_enable_button">Habilitar servicios de Google Play</string>
    <string name="common_google_play_services_update_title">Actualizar servicios de Google Play</string>
```

En la figura veiem un exemple de values-es/string.xml, com podem veure un string es defineix pel seu nom i pel seu valor. Els noms són comuns entre els idiomes, i són utilitzats com a ids, però internament no s'utilitza la cadena de caràcters com a id, sinó que es tradueixen a número hexadecimal, ja que utilitza un tipus long en lloc de string és molt més eficient i el desenvolupador no es dóna conte que en realitat està fent servir números hexadecimals.

El SDK d'Android crea un arxiu que fa com de diccionari entre la id com a cadena de caracters i la id com a tipus numèric. Aquest es troba a l'arxiu values/public.xml

```
<public type="string" name="k_two_factor" id="0x7f0a00a4" />
<public type="string" name="k_pull_twofactor" id="0x7f0a00a5" />
<public type="string" name="k_status_title" id="0x7f0a00a6" />
<public type="string" name="k_status_text_lock" id="0x7f0a00a7" />
<public type="string" name="k_status_text_lockbyparent" id="0x7f0a00a8" />
<public type="string" name="k_status_text_unlock" id="0x7f0a00a9" />
<public type="string" name="k_alllocked_title" id="0x7f0a00aa" />
<public type="string" name="k_alllocked_subtitle" id="0x7f0a00ab" />
<public type="string" name="k_schedule_diftimes" id="0x7f0a00ac" />
<public type="string" name="k_schedule_notconfigured" id="0x7f0a00ad" />
<public type="string" name="k_schedule_locked" id="0x7f0a00ae" />
<public type="string" name="x_actionbar_title" id="0x7f0a00af" />
```

Els valors hexadecimals que agafen les ids són arbitraris i no intuïtius. Per tant a l'hora de modificar la interfície xml i afegir nous ids, és complicat "fer-ho a mà" modificant directament els arxius xml, ja que és probable que fallem en la traducció a un número id.

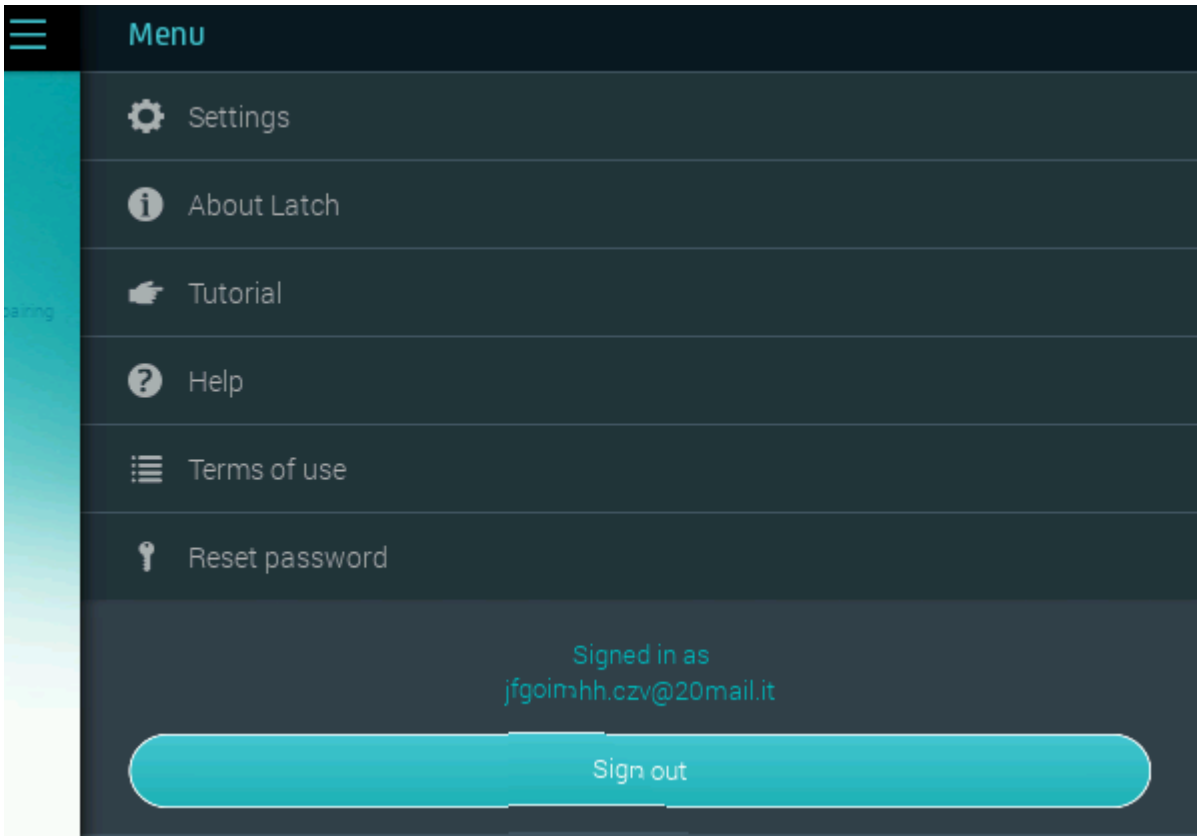
La solució senzilla a aquest problema és no fer servir noves ids, cosa realment complicada, ja que ens obliga a fer-ho tot per codi, i no podem aprofitar el potencial de les eines que ens ofereix l'sdk d'andoid.

He ideat però una nova tècnica per fer-ho, quan estem realitzant els canvis en la interfície, el que hem de fer és obrir un nou projecte Android (típicament amb eclipse) i copiar allà tota la carpeta res obtinguda amb apktool, i des de l'eclipse modificar de forma normal la interfície. En aquest moment quan nosaltres introduïm una nova id, internament eclipse ens farà tota la feina de fer la seva concordança en l'arxiu public.xml, de forma que no ens hem de preocupar. Una vegada hàgim fet tots els canvis desitjats a la interfície, simplement reemplacem la carpeta res produïda per apktool amb la del nostre nou projecte i utilitzem altra vegada apktool, aquest cop per empaquetar l'aplicació a un arxiu apk.

Aquesta nova tècnica ens alliberà del problema de crear noves ids, però hem de ser conscients d'aquesta traducció interna a l'hora de canviar el codi Jasmin, ja que encara que a l'eclipse utilitzem la cadena de caràcters (mitjançant la clase R generada pel propi eclipse), en codi jasmin fem servi el número hexadecimal.

4.3.2 Modificant l'interface Latch

Per configurar l'extensió, he decidit que el convenient és afegir un botó, en el menú de configuració de Latch. El qual iniciarà la meva activity.



Il·lustració 15: Menú Latch original

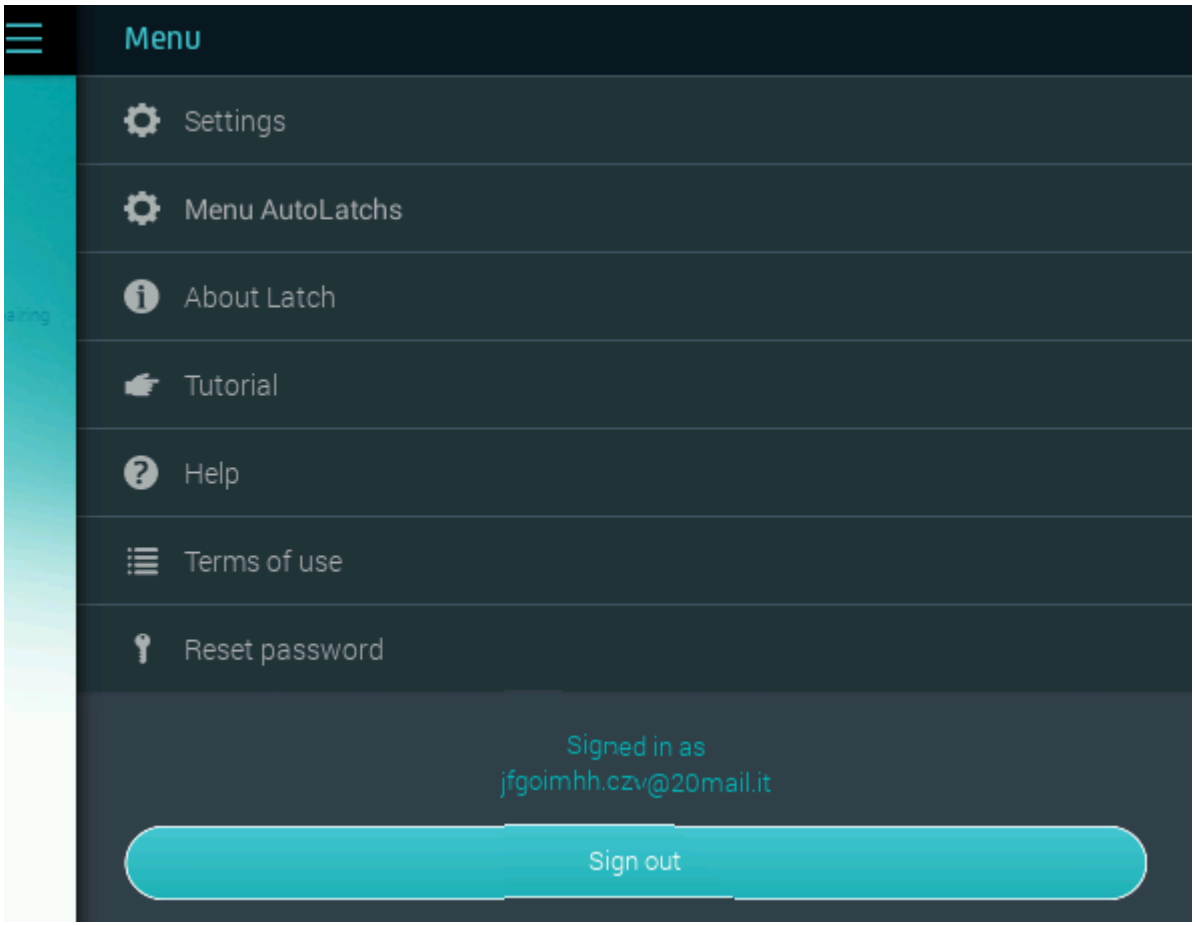
Per fer-ho primer he de saber quin és l'arxiu xml que defineix la interfície d'aquesta pantalla. Per trobar-ho ràpid busco l'string "About Latch" a res/values/strings.xml, i trobem:

```
<string name="nav_title">Menu</string>
<string name="nav_settings">Settings</string>
<string name="nav_about">About Latch</string>
<string name="nav_privacy">Terms of use</string>
<string name="nav_tutorial">Tutorial</string>
```

Si busquem ara per la id del string "nav_about", trobarem que l'arxiu que l'utilitza és res/layout/navigation_drawer.xml, però la seva versió d'id. És una pràctica habitual en el desenvolupament utilitzar el mateix nom per l'identificador de l'string(@string/nav_about) que per l'identificador de l'id del botó (@id/nav_about).

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:orientation="vertical" android:background="#f
  xmlns:android="http://schemas.android.com/apk/res/android">
    <LinearLayout android:gravity="center_vertical" android:orienta
        <TextView android:textSize="@dimen/action_bar_title_size" a
    </LinearLayout>
    <ScrollView android:background="#ff475663" android:layout_width
        <LinearLayout android:orientation="vertical" android:layout
            <Button android:textSize="@dimen/nav_button_text_size"
            <Button android:textSize="@dimen/nav_button_text_size"
            <Button android:textSize="@dimen/nav_button_text_size"
            <Button android:textSize="@dimen/nav_button_text_size"
            <Button android:textSize="@dimen/nav_button_text_size"
            <Button android:textSize="@dimen/nav_button_text_size"
            <ImageView android:layout_width="fill_parent" android:l
            <LinearLayout android:orientation="vertical" android:id
                <TextView android:textSize="@dimen/g_text_button_bi
                <TextView android:textSize="@dimen/g_text_button_bi
                <Button android:textSize="@dimen/g_text_button_bigg
            </LinearLayout>
        </LinearLayout>
    </ScrollView>
</LinearLayout>
```

Com podem veure es defineixen els 6 botons consecutivament, de manera que només he d'afegir el meu nou boto en el projecte que estic fent. Quedant la interfície de la següent manera:



Il·lustració 16: Menú Latch modificat

Però encara hem de programar el botó, per fer-ho busquem on estan programats els altres. Ja hem vist que l'identificador de l'id del botó és nav_about, busquem la seva correspondència a public.xml.

```
<public type="id" name="nav_title" id="0x7f060122" />
<public type="id" name="nav_settings" id="0x7f060123" />
<public type="id" name="nav_about" id="0x7f060124" />
<public type="id" name="nav_tutorial" id="0x7f060125" />
<public type="id" name="nav_help" id="0x7f060126" />
<public type="id" name="nav_privacy" id="0x7f060127" />
```

Com podem veure el seu valor és 0x7f060124, traduïm aquest nombre en decimal (el codi Jasmin utilitza números decimals), i el busquem en el codi Jasmin de l'aplicació. I trobem que el codi de programació del botó està a /ui/a.j:

```
ldc_w 2131099940
invokevirtual com/jeremyfeinstein/slidingmenu/lib/SlidingMenu/findViewById(I)Landroid/view/View;
checkcast android/widget/Button
astore 4
aload 4
ldc "fonts/roboto_light.ttf"
invokestatic com/elevenpaths/android/latch/h/d/a(Landroid/widget/Button;Ljava/lang/String;)V
aload 4
new com/elevenpaths/android/latch/ui/c
dup
aload 0
invokespecial com/elevenpaths/android/latch/ui/c/<init>(Landroid/app/Activity;)V
invokevirtual android/widget/Button/setOnClickListener(Landroid/view/View$OnClickListener;)V
```

Com podem veure, es busca la classe que defineix el botó mitjançant FindViewById i després s'utilitza h.d.a(Button, String) per configurar-lo visualment, després s'invoca setOnClickListener per assignar el codi que executarà quan es cliqui el botó.

Per programa el meu botó he d'afegir codi Jasmin en aquesta classe. Com he explicat anteriorment, la millor tècnica per fer-ho és no escriure tot un conjunt de codi Jasmin, sinó només invocar una funció estàtica, la qual programaré còmodament des de l'eclipse. Llavors només afegeixo les següents línies:

```
;Modificator.SetMenuBotton(this, localSlidingMenu);
aload 0
aload 1
invokestatic com/megacorp/autolatchplugin/Modificator/SetMenuBotton(Landroid/app/Activity;Landroid/view/View;)V
```

Com es veu crido a la meva funció Modificator.SetMenuBotton(Activty, View). Aquesta funció configura visualment el botó i llença la meva activity quan és clicat.

5. El projecte AutoLatchPlugin

Ja tenim modificat Latch per què cridi el meu projecte quan sigui necessari, ara toca, desenvolupar el meu projecte. El desenvolupament ara, és similar a desenvolupar qualsevol altra aplicació Android des de l'eclipse utilitzant l'sdk de Google, però sent conscients que ens hem d'adequar a Latch.

5.1 Model de dades i disseny de classes

Com hem vist en el punt 4.2.4, obtenim les operacions que té disponible l'usuari de l'aplicació. Les operacions[65] no són més que una abstracció jeràrquica dels serveis. Una operació es pot obrir o bloquejar, a més una operació pot tenir operacions filles, de manera que si l'operació pare està bloquejada, les operacions filles també ho estan.

És important, saber l'estructura de dades de les operacions, però no ens hem de preocupar sobre el seu control jeràrquic (familiar), ja que això es produeix des del servidor Latch. Tot i que l'usuari hauria de ser conscient de com funciona a l'hora d'utilitzar Latch i la nostra extensió.

També en el punt 4.2.4, es mostra que quan es reben les operacions de l'usuari, el meu projecte, omple un ArrayList amb la classe Operation. Aquesta classe conté la següent informació de l'operació, la seva id, la imatge que es mostra a Latch, el seu nom, i conte la Operació pare (null en cas que no tingui pare).

L'extensió de Latch, el que pretén és que quan un cert esdeveniment es produeix, una operació o varies siguin bloquejades o desbloquejades.

Amb aquest objectiu he creat la classe abstracta AutoOp, és l'encarregada de respondre amb un acció d'obertura/bloqueix d'una o varies operació, un esdeveniment rebut des del dispositiu mòbil. És abstracta, perquè és dependent del tipus d'esdeveniment al qual respon, els esdeveniments mostrats en aquest projecte són de posició GPS i de connectivitat Wireless.

AutoOp emmagatzema un nom que el descrigui per l'usuari, les id de les operacions que podria modificar, l'estat de l'esdeveniment i quina modificació ha de fer a les operacions segons l'estat de l'esdeveniment.

Per exemple, es pot configurar per un esdeveniment del tipus Wireless, amb l'estat de desconnexió i bloquejar l'operació login de Windows.

La classe AutoOp a més té la capacitat de serialitzarse i deserialitzarse en format JSON, per poder ser guardada en un string. Així l'usuari no ha de configurar l'extensió cada vegada que inicia el telefon mòbil o Latch és apagat.

El disseny utilitzat en la classe, com es pot intuir, està pensat perquè en cas que vulguem respondre a un altre esdeveniment del dispositiu, puguem desenvolupar el codi el més fàcil i ràpid possible.

Utilitzo també una classe auxiliar anomenada StaticData, principalment manté i guarda les dades de l'aplicació. Guarda en el dispositiu les dades de les classes AutoOp en format JSON utilitzant la classe SharedPreferences i és capaç de crear-les de nou quan s'inicialitza l'aplicació. Les dades que manté són principalment les operacions que té l'usuari de Latch.

5.2 Intercepció d'esdeveniments

Les aplicacions Android envien "missatges" a altres aplicacions o a la pròpia mitjançant una classe anomenada "Intent" [66] i per rebre'ls Android, utilitza la classe BroadcastReceiver[67].

Per informar el SO Android que tenim una classe preparada per rebre intents, podem utilitzar la funció Context.registerReceiver(), o utilitzant el tag <receiver> en el AndroidManifest.xml

Si volem rebre "events" del sistema operatiu en forma d'intents, hem d'especificar quins hem de rebre, utilitzant també AndroidManifest.xml però aquest cop utilitzant els tags <intent-filter> i <action>:

```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.megacorp.autolatchpluguin"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="21" />

    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
    <instrumentation android:name="com.megacorp.autolatchplugin.myClass" android:targetPackage="com.megacorp.autolatchplu
    <application android:allowBackup="true" android:label="@string/app_name" android:theme="@style/AppTheme">
        <receiver android:name="com.megacorp.autolatchplugin.AutoLatchReceiver">
            <intent-filter>
                <action android:name="android.net.conn.CONNECTIVITY_CHANGE"/>
                <action android:name="android.intent.action.BOOT_COMPLETED"/>
            </intent-filter>
        </receiver>
        <activity android:icon="@drawable/logo_dashboard" android:name="com.megacorp.autolatchplugin.ViewAutoLatchsActiv
        <activity android:icon="@drawable/logo_dashboard" android:name="com.megacorp.autolatchplugin.SelOperationsActivi
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity android:icon="@drawable/logo_dashboard" android:name="com.megacorp.autolatchplugin.SelEventActivity" an
        <activity android:icon="@drawable/logo_dashboard" android:name="com.megacorp.autolatchplugin.SelWifiActivity" and
    </application>
</manifest>
```

Com podem veure, en el AndroidManifest.xml del meu projecte, requerim la recepció de canvis en la connectivitat (esdeveniments Wireless) i rebre quan el SO s'ha iniciat.

També podem informar el SO que volem rebre un tipus d'intents per codi, com per exemple u fem per requerir els esdeveniments de geolocalització.

```
if(locManager==null){
    try{
        locManager = (LocationManager) context.getSystemService(android.content.Context.LOCATION_SERVICE);
        locManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 1000, 10f, this);
        locManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 1000, 10f, this);
    }catch(Exception e){/*Dispositiu sense GPS*/}
}
```

Requerim els intents mitjançant LocationManager.NETWORK_PROVIDER i LocationManager.GPS_PROVIDER, la diferència entre ells és que el primer quan el GPS es troba apagat ens dóna la geolocalització a partir de la posició que Google

té assignada a la xarxa Wirless en la qual estem connectats, i el segon ens dóna la posició obtinguda del gps del dispositiu [68].

El lector es pot preguntar, perquè requerim ser informats de quan el SO s'inicia, ho faig perquè així obligo al SO a executar la meva classe BroadcastReceiver i requerir per codi els esdeveniments de geolocalització que acabem de veure a més aprofito per iniciar la classe StaticData.

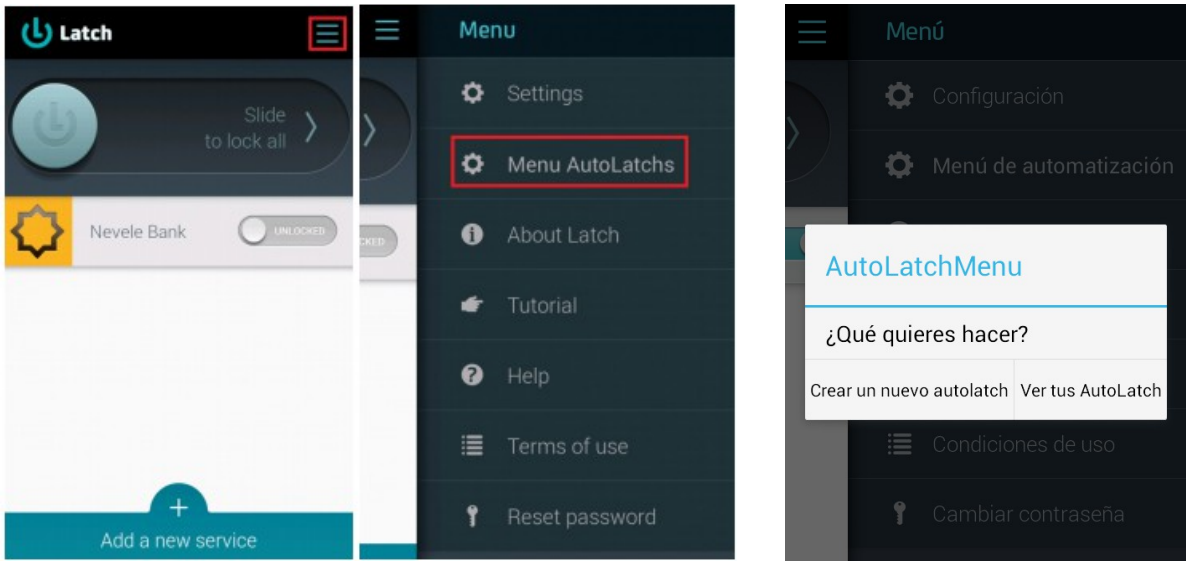
Quan es rep un "event" simplement s'itera entre totes les classes AutoOp mantingudes per StaticData i s'executa la seva funció newEvent.

5.3 Interfície d'usuari

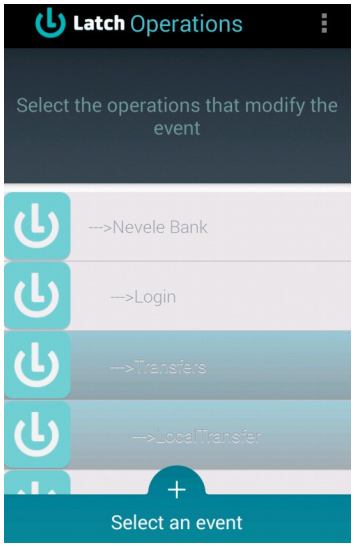
La interfície d'usuari del meu projecte, l'he fet amb una aparença el més semblant possible a la que te Latch, per tal que sembli una part de la mateixa aplicació i no un tros afegit; aconseguir una aparença similar m'ha estat fàcil, ja que abans he fet tot el procés d'analitzar l'aplicació.

La utilitat de la interfície consisteix a permetre a l'usuari que pugui configurar els seus AutoLatch fàcilment, i esborrar aquells que ja no vulgui utilitzar més.

Per això una vegada prem en el menú "Menu de automatización", es pregunta a l'usuari si vol crear nous AutoLatch o veure els ja creats, en cas que no en tingui cap de creat no es fa la pregunta, sinó que es passa directament en el menú de crear un AutLatch.

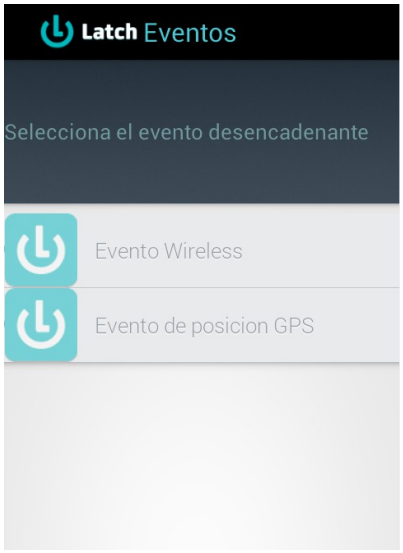


Per crear un AutoLatch en primer lloc es demana quines són les operacions que es vol que canviïn d'estat automàticament amb l'esdeveniment que es configurarà més endavant. Una vegada l'usuari, hagi seleccionat la o les operacions que vulgui, haura de prémer a "seleccionar un event".



Il·lustració 17: Menú de selecció d'operació

En aquesta versió, es pregunta si es vol escollir un esdeveniment Wireless o un esdeveniment GPS.



Il·lustració 18: Menú de selecció d'esdeveniment

Si se selecciona un esdeveniment Wireless, es llisten les xarxes Wireless segures a les quals ja hàgim accedit amb el telefono mòbil (Només es consideren xarxes Wireless segures aquelles amb password i que utilitzin com a mínim WPA).

Si en canvi se selecciona un esdeveniment GPS, se selecciona una zona geogràfica, tenint com a centre la posició on ens trobem i com a radi, els metres que configurem en el menú.

Una vegada l'esdeveniment ja sigues escollit, l'aplicació ens demanarà un nom perquè l'usuari pugui identificar aquest AutoLatch.

Després l'aplicació ens pregunta què es vol fer amb l'operació quant l'esdeveniment es dispari i per últim quan es vol que l'esdeveniment sigui reconegut com a disparat.

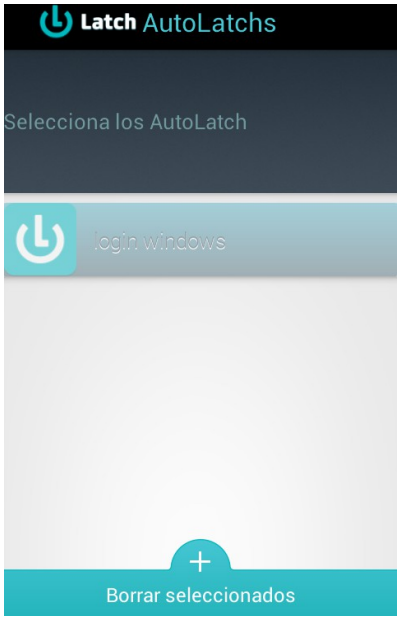
Exemples de configuració poden ser els següents:

- Tipus d'esdeveniment Red Wireless, Operació Bloquejar, Esdeveniment Sortir. Les operacions es bloquejaran quan l'usuari es desconnecti de la xarxa Wireless seleccionada o es connecti a alguna altra.

- Tipus d'esdeveniment GPS, Operació Desbloquejar, Event Entrar. Les operacions es desbloquejaran quan el telèfon intel·ligent rebi coordenades gps dins del radi geogràfic seleccionat.

- Tipus d'esdeveniment GPS, Operació Desbloquejar en entrar, bloquejar en sortir. Les operacions es bloquejaran quan el telèfon intel·ligent rebi coordenades fora de la zona geogràfica seleccionada i es desbloquejaran quan les rebi dins.

Quan seleccionem "veure els AutoLatch" es llisten els AutoLatch ja configurats i es poden esborrar aquells que seleccionem si premem al botó "Borrar seleccionats".



Il·lustració 19: Menú de AutoLatchs

6. Entorn de desenvolupament

Durant el desenvolupament és imperiós anar provant els canvis que es van fent en l'aplicació. El fet d'estar modificant una aplicació de tercers amb unes tècniques noves acabades d'idear, fa que no existeixi cap entorn de desenvolupament per treballar de forma lleugera.

Aquest fet podria provocar, que la creació de l'extensió fos molt farragosa, ja que moltes coses s'haurien de "fer a mà", i podria suposar un augment del temps utilitzat considerable o fins i tot que el projecte acabes en frustració.

Per això vaig idear un entorn que em facilites les tasques. En primer lloc, he de tenir organitzats els arxius amb els quals treballo de la següent manera:

OriginalApkDir : Directori on es troba el apk original.

AutoLatchPluguinDir : Directori on tinc el projecte que estendrà l'aplicació original

AutoLatchPluguinTempDir : Directori temporal

newManifest : Nou AndroidManifestMod.xml, ha de ser una mescla entre l'original i el que necessita l'extensió.

dex2jarToolsDir : Directori on es troba l'eina dex2jar

newLatchDir : Directori on es forma la nova aplicació

ModjasminCodeDir : Directori on tinc els arxius Jasmin que vull substituir pels originals.

AutoLatchPluguinAPK : Arxiu apk del meu projecte extensió, normalment autogenerat per eclipse.

adbExe : Eina adb del sdk d'Android

apktoolBat : el bat de l'eina apktool

Tenint organitzats els arxius i directoris d'aquesta manera, podem utilitzar els arxius bats que he creat, per forma la nova aplicació. Anem a veure com treballen, makeall.bat:

```
@echo off
set OriginalApkDir=F:\VBShare\latch2
set AutoLatchPluguinDir=F:\SpiderOak\eclipse\AutoLatchPluguin
set AutoLatchPluguinTempDir=F:\SpiderOak\uni\pfc\latch\AutoLatchPluguin
set newManifest=F:\VBShare\latch2\AndroidManifestMod.xml
set dex2jarToolsDir=f:\downloads\dex2jar-0.0.9.15\dex2jar-0.0.9.15
set newLatchDir=F:\SpiderOak\uni\pfc\latch\latchtoast
set ModjasminCodeDir=F:\SpiderOak\uni\pfc\latch\latchMOD\jasmincode
set AutoLatchPluguinAPK=F:\SpiderOak\eclipse\AutoLatchPluguin\bin\AutoLatchPluguin.apk
set adbExe=c:\android\adt-bundle-windows-x86_64-20130917\sdk\platform-tools\adb
set apktoolBat=apktoot.bat
@echo on

call copyResources.bat %OriginalApkDir% %AutoLatchPluguinDir%\res %newManifest% %dex2jarToolsDir% %apktoolBat%
call MakeLatchMod.bat %newLatchDir% %OriginalApkDir% %ModjasminCodeDir% %AutoLatchPluguinTempDir% %AutoLatchPluguinAPK% %adbExe% %dex2jarToolsDir%
```

Veiem que configuro les variables d'entorn segons l'acabat de descriure i llenço copyResources.bat i MakeLatchMod :

copyResources.bat

```
1  :: %1 = F:\VBShare\latch2
2  :: %2 = F:\SpiderOak\eclipse\AutoLatchPluguin\res
3  :: %3 = F:\VBShare\latch2\AndroidManifestMod.xml
4  :: %4 = f:\downloads\dex2jar-0.0.9.15\dex2jar-0.0.9.15
5  :: %5 = apktool
6  rd /S /Q %1\Latch
7  cd %1
8  call %5 d Latch.apk
9  xcopy %2 %1\Latch\res /S /Y
10 xcopy %3 %1\Latch\AndroidManifest.xml /Y
11 call %5 b Latch
12 call %4\d2j-apk-sign.bat -f -o %1\Latch\dist\LatchSigned.apk %1\Latch\dist\Latch.apk
```

copyResources, el que fa és utilitzar apktool per desempaquetar l'aplicació original, a continuació i posa els resources de la meva extensió i la torna a empaquetar. Finalment la signa per si es vol provar la nova interfície generada.

MakeLatchMod.bat

```
::%1 = F:\SpiderOak\uni\pfc\latch\latchtoast
::%2 = F:\VBShare\latch2
::%3 = F:\SpiderOak\uni\pfc\latch\latchMOD\jasmincode
::%4 = F:\SpiderOak\uni\pfc\latch\AutoLatchPluguin
::%5 = F:\SpiderOak\eclipse\AutoLatchPluguin\bin\AutoLatchPluguin.apk
::%6 = c:\android\adt-bundle-windows-x86_64-20130917\sdk\platform-tools\adb
::%7 = f:\downloads\dex2jar-0.0.9.15\dex2jar-0.0.9.15
rd /S /Q %1
mkdir %1
ECHO F | xcopy %2\Latch\dist\LatchSigned.apk %1\Latch.apk /Y
call %7\d2j-dex2jar.bat -f -o %1\newjar.jar %1\Latch.apk
call %7\d2j-jar2jasmin.bat -f -o %1\jasmincode %1\newjar.jar
xcopy %3 %1\jasmincode /S /Y
call copyAutoPluguin.bat %4 %5 %1 %7
call RepackLatch.bat %1 %7
@echo off
echo to do
echo      add %1\classes.dex to %1\newLatch.apk
pause
@echo on
call SignLatch.bat %1 %7
call InstallLatch.bat %1 %6
```

MakeLatchMod, utilitza dex2jar per obtenir el codi Jasmin, després i copia el meu codi Jasmin modificat de l'original (per exemple l'explicat en el 4.2.4), copyAutoPluguin.bat fa una cosa similar, obté el codi Jasmin del projecte d'extensió i hi copia aquest a la mescla anterior:

copyAutoPluguin.bat

```
1  :: %1 = F:\SpiderOak\uni\pfc\latch\AutoLatchPluguin
2  :: %2 = F:\SpiderOak\eclipse\AutoLatchPluguin\bin\AutoLatchPluguin.apk
3  :: %3 = F:\SpiderOak\uni\pfc\latch\latchtoast
4  :: %4 = f:\downloads\dex2jar-0.0.9.15\dex2jar-0.0.9.15
5  rd /Q /S %1
6  mkdir %1
7  ECHO F | xcopy %2 %1\AutoLatchPluguin.apk /Y
8  call %4\d2j-dex2jar.bat -f -o %1\newjar.jar %1\AutoLatchPluguin.apk
9  call %4\d2j-jar2jasmin.bat -f -o %1\jasmin %1\newjar.jar
10 xcopy %1\jasmin\com %3\jasmincode\com /S /Y
```

RepackLatch, el que fa és agafar tot aquest projecte amb la mescla dels 3 codis Jasmin i construir un nou jar, verificant a més que el nou codi generat és correcte i extreu l'arxiu classes.dex

RepackLatch.bat

```
1  :: %1 = F:\SpiderOak\uni\pfc\latch\latchtoast
2  :: %2 = f:\downloads\dex2jar-0.0.9.15\dex2jar-0.0.9.15
3  ::
4  call %2\d2j-jasmin2jar.bat -f -o %1\repack.jar %1\jasmincode
5  call %2\d2j-asm-verify.bat %1\repack.jar
6  call %2\d2j-jar2dex.bat -f -o  %1\classes.dex %1\repack.jar
7  ECHO F | xcopy %1\Latch.apk %1\newLatch.apk /Y
```

En aquest punt (MakeLatchMod línies 16-20) demana que posem l'arxiu classes.dex en l'arxiu newLatch.apk, això no ho faig automàticament per línia de comandes, ja que el programa per fer-ho és de pagament (gestor d'arxius zip) i jo n'utilitzava un de gratuït que no te línia de comandes.

Finalment es signa el nou apk, i s'instal·la utilitzant adb.

Com es pot veure utilitzo diversos arxius bat, això és per si vull fer només una part de les tasques mentre estic desenvolupant. Però si vull generar i instal·lar en el dispositiu el meu projecte estès, només cal una crida a makeall.bat, cosa que fa la feina molt més lleugera.

7. Planificació i costos

En aquest apartat es plasma els recursos utilitzats per realitzar tot el present projecte fi de carrera.

7.1 Planificació temporal

El projecte va començar el 29 de setembre de 2014 i es va donar per acabat durant la segona setmana de juliol del 2015. Encara que pel que fa a l'aprenentatge sempre he estat interessat pel món de l'enginyeria inversa i *cracking*, llegint llibres o resolent exercicis proposats del tipus crackme. No es pot establir un període de temps exacte per aquesta afició personal, però sí que el podria valorar en com a mínim 200 hores.

En un principi invertia entre 6 i 8 hores diàries, entre la tarda i la nit, després per realitzar la memòria utilitzava 2 hores setmanals i per juliol vaig tornar a passar a les 8 hores diàries.

29 de Setembre 2014 - 6 d'Octubre 2014: Escollir aplicació, buscar debilitats i possibles millores. 6 dies x 8 = 48 hores

7 d'Octubre 2014 - 31 d'Octubre: Aprenentatge sobre l'enginyeria inversa en aplicacions Android, posada en pràctica i cerca de noves tècniques. 18 dies x 8 = 144 hores

3 de Novembre 2014 - 18 de Novembre 2014: Enginyeria inversa de Latch i proves de manipulació. 12 dies x 8 = 96 hores

19 de Novembre 2014 - 12 de Desembre 2014: Desenvolupament i acomplament de l'extensió per Latch. 19 dies x 6 = 114 hores

15 de Desembre 2014 - 24 de Desembre 2015 : Període de proves de l'extensió. 8 dies x 8 = 64

7 de Gener 2015 - 8 de Gener 2015 : Documentació de la nova extensió de Latch. 2 dies x 8 = 16 hores

19 de Gener 2015 - 8 de Juny 2015 : Realització de la memòria. 19 setmanes x 2 + 6 dies x 8 = 86 hores

Total hores = 200+48+144+96+114+64+16+86 = **752 hores**

7.2 Costos materials

Bàsicament s'exposa quins recursos han hagut de ser adquirits per la producció. Cal aclarir que pel que fa el software utilitzat, s'ha prioritzat el software lliure, per raons filosòfiques de l'autor i econòmiques.

Recurs	Tipus	Preu
Ordinador	Hardware	950 €
Samsung Galaxy SII	Hardware	399 €
Windows 7	Software	Inclòs a l'ordinador
Router Wifi	Hardware	35,95 €
Internet	Servei	19,95 €
Eclipse	Software	Gratuït
SDK Android	Software	Gratuït
Eines explicades	Software	Gratuït
Eines GoogleDocs	Software	Gratuït

Cost material = 950+399+35,95+19,95*10 = **1.584,45 €**

7.3 Recursos humans i cost total

Ja que he tingut en la planificació he tingut en compte el temps de formació d'un enginyer informàtic acabat de sortir de la facultat, i que la documentació del projecte és força tècnica i per tant està fent la mateixa tasca que el desenvolupador, he establert un preu/hora únic, de 13,51 €/hora.

El preu de 13,51 €/hora ha estat establert en base que suposaria un cost per l'empresa de 2702 €/mes, tenint en compte, la nòmina bruta mensual (1.635 €), seguretat social (540 €), desgast material (50 €), assegurança de conveni (15 €), prevenció per acomiadament o baixa per malaltia (272 €) i impacte vacacional (190 €).

Proporcionant uns ingressos de 1200 €/mes nets pel treballador, que és el que l'oferta de treball ofereix per mitjana als recents estudiants sortits de la FIB.

Cost recursos humans = 752 * 13,51 = **10.159,52 €**

Cost total del projecte = 10.159,52 + 4.512 **= 14.671,52 €**

8. Conclusions

Els objectius del projecte han sigut realitzats exitosament. Aconseguint una nova aplicació a partir d'una altre creada per tercers i estesa per mi.

S'ha pogut realitzar utilitzant només el paquet d'instal·lació apk obtingut de Google Play, i tot i que l'aplicació havia estat prèviament ofuscada utilitzant ProGuard[69] o una eina molt similar.

Hem vist que les eines open source disponibles tot hi semblar rudimentàries són suficients per porta a terme els objectius, i que si a més creem un entorn de desenvolupament correcte, es pot automatitzar l'ús d'aquestes eines i podem tenir un entorn de treball eficient.

La consecució dels objectius posen de manifest alguns problemes de seguretat en el món Android. El codi font, així com els recursos de l'aplicació estan exposats a qualsevol investigador, si es pot analitzar el codi font, es poden buscar debilitats en el codi o fins i tot copiar-lo i crear una nova aplicació que l'utilitzi.

Un escenari probable, podria ser que un desenvolupador malintencionat observi aplicacions d'èxit en algun market d'Android. Podria descarregar una d'elles, fer alguns canvis visuals i pujar-la amb el seu nom. Amb relativament poc esforç, el desenvolupador malintencionat podria quedar-se part dels guanys dels desenvolupadors originals.

Per altra banda la meva aplicació només buscava millorar l'original i aportar un afegit als usuaris, tenia el consentiment dels creadors de Latch, fins i tot es va reconèixer públicament la feina realitzada. Però algú malintencionat en lloc de busca una millora podria afegir-hi codi maliciós, per exemple en el cas de Latch, fer que es puguin controlar els serveis remotament per un altre usuari, i aconseguir que els usuaris descarreguin aquesta nova aplicació maliciosa, de forma que seria pràcticament impossible que l'usuari veies que no utilitza l'aplicació original.

Cal remarcar que això no només passa amb l'aplicació Latch sinó que pràcticament amb qualsevol aplicació Android. Per exemple es podria modificar WhatsApp perquè envies totes les conversacions a cibercriminals.

Finalment, vull expressar que a escala personal la realització d'aquest projecte ha suposat una experiència enriquidora, m'ha permès explorar les aplicacions Android a un nivell més detallat i tenir una visió més amplia del SO. A més aquest projecte m'ha permès ser creatiu tant a l'hora de buscar millores per Latch, com a l'hora de buscar noves tècniques per analitzar i estendré aplicacions.

Bibliografia

1: , Android, , https://www.android.com
2: , Latch. El interruptor de seguridad para tu vida digital, , https://latch.elevenpaths.com
3: , WordPress.com: Create a free website or blog, , https://es.wordpress.com
4: , Autenticación de dos factores, , https://www.securingthehuman.org/newsletters/ouch/issues/OUCH-201211_sp.pdf
5: , Latch Plugins and SDKs, , https://latch.elevenpaths.com/www/plugins_sdks.html
6: , Phishing, , https://en.wikipedia.org/wiki/Phishing
7: , A Guide to Sniffing Out Passwords and Cookies, , http://lifehacker.com/5853483/a-guide-to-sniffing-out-passwords-and-cookies-and-how-to-protect-yourself-against-it
8: , Sniffing Passwords Over a Wifi Connection [Linux/Backtrack5], , http://www.hackavision.com/2011/07/sniffing-passwords-over-wifi-connection.html
9: , Hacking the WordPress Login by Capturing WordPress Usernames and Passwords, , http://www.wpwhitesecurity.com/wordpress-security/hacking-wordpress-login-capturing-usernames-passwords/
10: , DNS Hijacking: What is it and How it Works, , http://www.gohacking.com/dns-hijacking/
11: , DNS Hacking (Beginner to Advanced), , http://resources.infosecinstitute.com/dns-hacking/
12: , Acceso a un router remoto. Historia de un ataque MITM, , http://www.hacking-etic.cat/?p=326&lang=es
13: , Exploit (computer security), , https://en.wikipedia.org/wiki/Exploit_(computer_security)
14: , Malware, , https://en.wikipedia.org/wiki/Malware
15: , Linux.com | The source for Linux information, , https://www.linux.com
16: , Linux, , https://en.wikipedia.org/wiki/Linux
17: , Open Handset Alliance, , http://www.openhandsetalliance.com
18: , iOS, , https://en.wikipedia.org/wiki/IOS
19: , Windows Phone, , https://en.wikipedia.org/wiki/Windows_Phone
20: , Android Security, , https://source.android.com/devices/tech/security/
21: , How Secure Is Android, Really? - Lifehacker, , http://lifehacker.com/how-secure-is-android-really-1446328680
22: , Arquitectura De Android: ¿Como Funciona Este SO?, , http://www.hermosaprogramacion.com/2014/08/android-app-funciona-como/
23: , Java, , https://www.java.com/
24: , What is…the Dalvik virtual machine?, , http://www.electronicsweekly.com/eyes-on-android/what-is/the-dalvik-virtual-machine-2011-10/
25: , Android SDK, , https://developer.android.com/sdk/index.html
26: , Android software development, , https://en.wikipedia.org/wiki/Android_software_development
27: , Android NDK, , https://developer.android.com/tools/sdk/ndk/index.html
28: , Welcome to App Inventor for Android!, , http://code.google.com/p/app-inventor-for-android/
29: , Android application package, , https://en.wikipedia.org/wiki/Android_application_package
30: , Google Play, , https://play.google.com
31: , Allow app installs from 'unknown sources' | Android Central, , http://www.androidcentral.com/allow-app-installs-unknown-sources
32: , Zip (file format), , https://en.wikipedia.org/wiki/Zip_(file_format)

33: , A deep dive into DEX file format, , http://events.linuxfoundation.org/sites/events/files/slides/ABS2014.pdf
34: , App Manifest, , http://developer.android.com/guide/topics/manifest/manifest-intro.html
35: , Descriptions of Android's Resources.arsc, , https://ekasiswanto.wordpress.com/2012/09/19/descriptions-of-androids-resources-arsc/
36: , Dalvik opcodes, , http://pallergabor.uw.hu/androidblog/dalvik_opcodes.html
37: , JASMIN USER GUIDE, , http://jasmin.sourceforge.net/guide.html
38: , Extensible Markup Language (XML), , http://www.w3.org/XML/
39: , Tutorial: Understanding Android App Signing, , https://coronalabs.com/blog/2014/08/26/tutorial-understanding-android-app-signing/
40: , Reverse engineering, , https://en.wikipedia.org/wiki/Reverse_engineering
41: , Beginners Guides: Understanding and Creating Batch Files, , http://www.pcstats.com/articleview.cfm?articleID=1767
42: , Linux Shell Scripting Tutorial v1.05r3A Beginner's handbook, ,
43: , Chapter 4. The class File Format, , https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-4.html
44: , Graphical user interface, , https://en.wikipedia.org/wiki/Graphical_user_interface
45: , JAR (file format), , https://en.wikipedia.org/wiki/JAR_(file_format)
46: , Eclipse IDE for Java Developers, , www.eclipse.org
47: , Static Methods, , http://introcs.cs.princeton.edu/java/21function/
48: , Understanding the Android File Hierarchy, , http://www.all-things-android.com/content/understanding-android-file-hierarchy
49: , Getting Started with Android Forensics, , http://resources.infosecinstitute.com/getting-started-android-forensics/
50: , Android Debug Bridge, , http://developer.android.com/tools/help/adb.html
51: , Using Hardware Devices, , http://developer.android.com/tools/device.html
52: , How To Enable USB Debugging On Your Android Phone, , http://www.groovypost.com/howto/mobile/how-to-enable-usb-debugging-android-phone/
53: , Obfuscation (software), , https://en.wikipedia.org/wiki/Obfuscation_(software)
54: , Protect Your Java Code — Through Obfuscators And Beyond, , http://www.excelsior-usa.com/articles/java-obfuscators.html
55: , Understanding HTTP Basics, , http://learn.onemonth.com/understanding-http-basics
56: , Understanding SOAP and REST Basics, , http://blog.smartbear.com/apis/understanding-soap-and-rest-basics/
57: , JSON, , https://en.wikipedia.org/wiki/JSON
58: , Static program analysis, , https://en.wikipedia.org/wiki/Static_program_analysis
59: , Dynamic program analysis, , https://en.wikipedia.org/wiki/Dynamic_program_analysis
60: , Dynamic Analysis vs. Static Analysis, , https://software.intel.com/sites/products/documentation/doclib/iss/2013/inspector/lin/ug_docs/GUID-E901AB30-1590-4706-94B1-9CD4736D8D2D.htm
61: , Stack trace, , https://en.wikipedia.org/wiki/Stack_trace
62: , Android Activity, , http://developer.android.com/reference/android/app/Activity.html
63: , Take an in-depth look at the Java Reflection API, , http://www.javaworld.com/article/2077015/java-se/take-an-in-depth-look-at-the-java-reflection-api.html

64: , Resource Types, , http://developer.android.com/guide/topics/resources/available-resources.html
65: , Latch: Cómo proteger las identidades digitales, , http://www.elladodelmal.com/2013/12/latch-como-proteger-las-identidades_16.html
66: , Intent | Android Developers, , http://developer.android.com/reference/android/content/Intent.html
67: , BroadcastReceiver | Android Developers, , http://developer.android.com/reference/android/content/BroadcastReceiver.html
68: , LocationManager | Android Developers, , http://developer.android.com/reference/android/location/LocationManager.html
69: , ProGuard, , proguard.sourceforge.net/